|  |  |  |  |
| --- | --- | --- | --- |
|  | | для прик эмбл |  |
| *МИНОБРНАУКИ РОССИИ* | | | |
| ***Федеральное государственное бюджетное образовательное учреждение***  ***высшего профессионального образования***  *"Московский* *технологический университет*  МИРЭА | | | |
| Институт информационных технологий | | | |
| *(наименование факультета)* | | | |
| Математического обеспечения и стандартизации информационных технологий | | | |
| *(наименование кафедры)* | | | |

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| по дисциплине | |
| «Объектно – ориентированное программирование» | |
| *(наименование дисциплины)* | |
| Тема курсового проекта (работы) **«Проектирование и реализация**  *(наименование темы)*  **программной системы «Автомат по продаже кофе»** | |
|  | |
| Студент группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(учебная группа)* | *Фамилия И.О* |
| Руководитель курсового проекта (работы)*должность, звание, ученая степень* | *Фамилия И.О* |
| Рецензент (*при наличии*)  *должность, звание, ученая степень* | *Фамилия И.О* |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2018г. | *(подпись студента)* |
|  |  |  |
| «Допущен к защите» | «\_\_»\_\_\_\_\_\_\_2018 г. | *(подпись руководителя)* |

Москва 2018

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *МИНОБРНАУКИ РОССИИ* | | | | |
| ***Федеральное государственное бюджетное образовательное учреждение***  ***высшего профессионального образования***  *"Московский технологический университет*  МИРЭА | | | | |
| Институт информационных технологий | | | | |
| *(наименование факультета)* | | | | |
| Математического обеспечения и стандартизации информационных технологий | | | | |
| *(наименование кафедры)* | | | | |
|  | |  | | |
|  | |  | | |
|  | |  | | |
| **ЗАДАНИЕ** | | | | |
| на выполнение курсовой работы | | | | |
| **по** **дисциплине** «Объектно – ориентированное программирование» | | | | |
|  | | | | |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Группа\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |
| 1. Тема Проектирование и реализация программной системы «Автомат по продаже кофе» | | | | |
| 1. Исходные данные: | | | | |
| 2.1 Функционал, определённый поставленной задачей | | | | |
| 2.2 Среда разработки (IDE) Microsoft Visual Studio, язык программирования C++ | | | | |
| 2.3 Использовать обозначения языка UML | | | | |
| * 1. Перечень вопросов, подлежащих разработке, и обязательного графического материала: | | | | |
| 3.1 Разработать класс «Coffee» | | | | |
| 3.2 Определить список функций программы | | | | |
| 3.3 Разработать диаграмму классов | | | | |
| 3.4 Построить концептуальную модель предметной области | | | | |
| 3.5 Разработать графический интерфейс пользователя | | | | |
| 3.6 Выполнить тестирование программной системы | | | | |
| 1. Срок представления к защите курсовой работы: до «\_\_\_» \_\_\_\_\_\_\_201\_ г. | | | | |
|  | | | | |
| Задание на курсовой  проект, (работу) выдал | «\_\_\_»\_\_\_\_\_\_201\_\_г. | |  | *Ф.И.О. руководителя*  *проекта* |
| Задание на курсовой  проект, (работу) получил | «\_\_\_»\_\_\_\_\_\_201\_\_г. | |  | *Ф.И.О. студента -*  *исполнителя*  *проекта* |

Оглавление

[Реферат 4](#_Toc516157238)

[Введение 5](#_Toc516157239)

[1 Теоретическое введение 7](#_Toc516157240)

[2 Проектная часть 8](#_Toc516157241)

[2.1 Постановка задачи 8](#_Toc516157242)

[2.2 Проектное решение 8](#_Toc516157243)

[2.3 Структура пользовательского интерфейса 12](#_Toc516157244)

[3 Экспериментальная часть 13](#_Toc516157245)

[3.1 Описание подхода к тестированию программы 13](#_Toc516157246)

[3.2 Тестирование 14](#_Toc516157247)

[3.3 Инструкция пользователя 18](#_Toc516157248)

[Заключение 22](#_Toc516157249)

[Список литературы 23](#_Toc516157250)

[Приложения 24](#_Toc516157251)

# Реферат

Представленная работа содержит 30 страниц 9 рисунков, 5 таблиц, 8 разделов.

Введение знакомит с актуальностью рассматриваемой темы, даёт краткую историческую справку по развитию информационных технологий вообще и языка UML – в частности.

Теоретическое введение кратко описывает особенности моделируемой предметной области.

Проектная часть содержит более подробный анализ предметной области, её декомпозицию на сущности и выявление связи между ними. Кроме того, в этом разделе производится проектирование разрабатываемой системы и приводится детальное описание графического интерфейса пользователя.

В экспериментальной части разрабатывается подход к тестированию системы, определяется набор тестов и отражаются результаты проведённого тестирования. В этом разделе также содержится подробное описание последовательности действий пользователя по работе с системой.

В заключении подводятся итоги проделанной работы.

Список использованных источников отражает степень охвата теоретического материала в работе.

Приложения содержат листинги модулей разработанной программной системы.

# Введение

В настоящее время, с развитием научно-технического прогресса и информационных технологий, сложность автоматизируемых предметных областей постоянно и неуклонно возрастает. Программист проводит всё своё рабочее время, по сути, в борьбе со сложностью программ, отлаживая свой код в поисках допущенных ошибок. Ради борьбы со сложностью эволюционируют языки программирования. После машинных кодов и перфокарт появились текстовые редакторы и язык ассемблера, что позволило писать чуть более сложные и объёмные программы при меньших трудозатратах. Далее появились процедурные языки высокого уровня - C, Паскаль и многие другие, что перевело технологии программирования на новый уровень.

Следующим крупным скачком на пути борьбы со сложностью стало появление объектно-ориентированного подхода и разработка соответствующих языков и компиляторов. Так C эволюционировал в C++. Но прогресс и сопутствующий рост сложности информационных систем не останавливаются на месте и вот появляются всё новые подходы к проектированию программ. Акцент при разработке сложных программных систем смещается с собственно написания кода в сторону проектирования и моделирования, что неизбежно приводит к совершенствованию соответствующих методик. При проектировании всегда возникает вопрос, как перенести на бумагу и электронной носитель идеи, которые возникают в голове разработчика - а главное, чтобы донести эти идеи в понятной форме до коллег.

К концу прошлого века сформировалась критическая масса различных подходов к анализу и разработке сложных информационных систем. Эволюция различных подходов привела к осознанию необходимости объединения их в один, чтобы совместными усилиями развивать и совершенствовать универсальный подход с моделированию предметной области. В результате появился универсальный язык моделирования, UML (англ. Unified Modeling Language). В 1990 году была представлена его официальная версия 1.0. В настоящее время эволюция UML также не стоит на месте - в августе 2005 года опубликована формальная спецификация 2.0, а в июне 2015 года - версия 2.5.

Данная работа посвящена изучению современных подходов к разработке сложных информационных систем, начиная от их моделирования и заканчивая разработкой готового продукта на языке высокого уровня и тестированием. В качестве языка программирования выбран С++ и его современный компилятор от компании Microsoft.

Целью работы является получение практических навыков по проектированию и реализации программной системы с применением объектно-ориентированного подхода.

В качестве предметной области для демонстрации возможностей языков UML и C++ выбрано функционирование автомата по приготовлению кофейных кофе. В результате данной работы будет спроектирована модель предметной области и разработана программа с графическим пользовательским интерфейсов, функционирование которой будет имитировать для пользователя работу реального физического устройства.

# Теоретическое введение

Предметная область разрабатываемой программной системы – автомат по продаже кофе. Как известно, такие автоматы позволяют внести сумму денег и заказать понравившийся тип напитка – например, кофе «Эспрессо» или «Каппучино». В случае превышения стоимости напитка, автомат выдаёт сдачу. Очевидно, что напитки, купюры и монеты в этом аппарате ограничены, и через какое-то время работы он может начать отказывать в выдаче ряда напитков в виду отсутствия сдачи или сырья для приготовления самих напитков (вода, сахар, кофейные зёрна или капсулы).

Как известно, модель – это упрощённое представление системы реального мира, совпадающее с оригиналом по какому-либо набору параметров или с учётом ряда допущений. В нашей программной реализации, которая будет моделировать поведение реального кофейного автомата, мы тоже применим некоторые допущения. Например, напитки будут храниться не в виде сырья, а в виде условных порций, готовых к употреблению. При этом ограничения на количество внутренних ресурсов остаются в силе и могу с течением времени заканчиваться.

# Проектная часть

## Постановка задачи

В ходе данной работы необходимо разработать программную систему, моделирующую функционирование автомата по продаже кофейных напитков. Необходимо использовать при проектировании системы язык моделирования UML, среду разработки Microsoft Visual Studio и язык программирования C++.

Представим эту устройство как некую систему, в памяти которой хранятся два набора объектов – порции кофейных напитков нескольких типов, а также купюры и монеты различных номиналов. На вход устройства поступает пара запросов: предоплата в виде определённой денежной суммы и выбранный пользователем тип кофейного напитка. В случае наличия в моделируемом устройстве порции напитка выбранного типа и наличных денег для сдачи, эта пара запросов успешно обслуживается. В противном случае предоплата не принимается и выдаётся сообщение об ошибке.

## Проектное решение

Основная сущность, которая очевидным образом выделяется из предметной области – это кофе, а точнее, кофейный напиток. В самом общем случае, кофейные напитки бывает разных видов («Капучино», «Эспрессо», «Американо» и т.д.) и могут иметь разную цену в зависимости от вида. Также общеизвестно, что кофе может быть растворимым, зерновым и молотым. Для разрабатываемой модели часть свойств предметной области придется опустить, чтобы не делать модель перегруженной неиспользуемыми деталями. В нашем случае сущность «Кофе» будет подразумевать кофейный напиток конкретного вида и иметь такие базовые характеристики как собственно «Вид» и «Цена», остальное нас пока не интересует.

Далее, следует иметь в виду, что автомат по производству кофе выдаёт напитки, как правило, стаканчиками фиксированного объёма – иначе говоря, порциями. Именно порции кофе мы будем готовить и учитывать. При этом видится нецелесообразным выделять отдельную сущность «Порция кофе», поскольку мы тогда получим избыточное количество достаточно однотипных экземпляров, которые в нашей модели не будут обладать какими-то конкретными характеристиками. Действительно, такие характеристики предметной области реального мира как, например, «Объём», «Температура» и другие, заранее определены и всегда одинаковы для всех экземпляров. Отличаться они могут только признаком «Вид», который уже присутствует в упомянутом ранее старшем типе «Кофе» (кофейный напиток).

Учитывая изложенные соображения, сущность «Кофе» не будет подразумевать конкретную порцию – вместо этого мы добавим к ней характеристику «Количество», которая и будет учитывать всё множество имеющихся в наличии порций напитка данного вида. Операции по приготовлению выделить не представляется возможным, поскольку эти детали предметной области выходят за рамки рассмотрения (исходя из поставленной задачи, управления реальным кофейным автоматом не предусматривается). В противном случае имело бы смысл выделить каждый вид кофе в отдельную сущность, наследующую от базовой абстракции. Впрочем, для удобства можно обозначить вспомогательные операции инкремента (загрузить в автомат кофейные напитки) и декремента (забрать) количества, а также методы доступа и чтения атрибутов. Представлять сущность «Кофейный напиток» будет класс **Coffee** нашей программной системы.

Кроме напитков, кофейный автомат хранит также банкноты и монеты, используемые для выдачи сдачи пользователю в случае превышения предоплаченной суммой стоимости выбранного напитка. На первый взгляд, возникает сущность «Деньги», которая может объединить в себе и банкноты, и монеты, и имеет такие отличительные признаки как «Номинал», «Тип» (банкнота или монета) и количество. На самом деле, для решения задач виртуальной кофейной машины, эта сущность, с ее характеристиками, является скорее избыточным информационным шумом – ведь для учёта имеющихся в наличии денежных средств и правильного расчёта выдаваемой сдачи достаточно иметь множество пар «номинал»-«количество», даже не различая, монета это или банкнота (в реальности они практически не пересекаются). С этой задачей прекрасно справляется ассоциативный контейнер стандартной библиотеки языка C++, **std::map**.

Примем за условие, что кофейный автомат работает исключительно в рублёвой зоне, тогда мы можем утверждать, что знаем заранее все возможные номиналы банкнот и монет. Предположим, что наш кофейный автомат из соображений безопасности работает с номиналами до 500 руб. включительно. Тогда мы можем определить перечисление «Номинал» - назовём его **enum** **CashValue** - и установить его в качестве ключа нашего денежного «словаря» (ассоциативного контейнера). В качестве значений этого словаря будет выступать количество соответствующих денежных знаков. В программе определим тип **typedef std::map<CashValue, int> CashAvail** и член класса этого типа – **сashAvail**, который будет выполнять роль контейнера денежных знаков.

Для расчёта сдачи с внесённой суммы будем использовать жадный алгоритм, то есть сперва проверим, сколько самых крупных купюр «помещается» в сумму сдачи, затем следующих по номиналу и т.д., до самых мелких. Использование жадного алгоритма даст нам наименьшее количество денежных знаков, которые нужно потратить на покрытие всей суммы сдачи. Это обусловлено тем, что в российской системе денежных знаков каждый следующий по возрастанию номинал крупнее предыдущего в два или более раза. В противном случае пришлось бы прибегать к методам динамического программирования.

Следующая сущность нашей системы – это, очевидно, сам автомат по производству кофе. Это сущность будет инкапсулировать в себе все детали операций по приготовлению кофе. Пользователь сможет взаимодействовать с этой сущностью через лаконичный интерфейс, содержащий основные операции «ПросмотрАссотримента» и «ПриготовитьКофе». Реализуем эту сущность с помощью класса **CoffeeMashine**.

Сущности разрабатываемой программной системы изображены в виде диаграммы классов UML, представленной на рисунке 1.

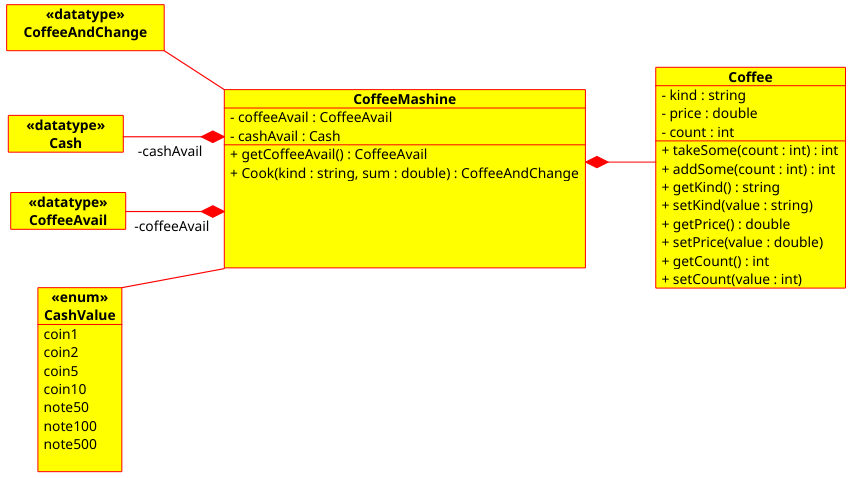


Рисунок 1. Диаграмма классов разрабатываемой системы

Наилучшим и наиболее приближенным к реальности отношение сущностей «Кофе» и «Кофейный автомат» принята композиция. В программной реализации кофейные напитки будут храниться, как и наличные, в соответствующем словаре. В качестве ключа будет использован атрибут «Вид» строкового типа, а в качестве значения – сущность «Кофе». В программе определим тип **typedef std::map<std::string, Coffee> CoffeeAvail** и член класса этого типа – **сoffeeAvail**, который будет выполнять роль контейнера кофейных напитков.

В качестве клиента класса **CoffeeMashine** будет выступать основная программа с графическим интерфейсом, которая будет через элементы управления транслировать пользователю программы управление кофейным автоматов.

## Структура пользовательского интерфейса

Главным окном приложения является диалоговое окно с текстовым полем для указания вносимой денежной суммы, кнопками приготовления кофе и выхода, а также списками ассортимента и результатов. Форме главного диалога присвоен идентификатор mainWnd, его вид в режиме конструктора изображён на рисунке 2.

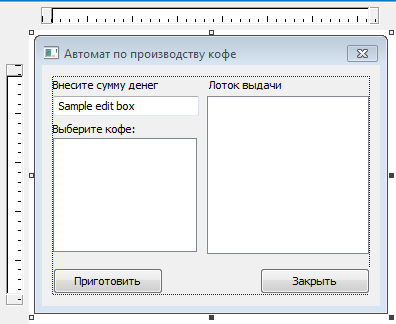


Рисунок 2. Макет главного окна приложения

Список элементов управления главного окна представлен в таблице 1.

Таблица 1. Элементы управления

|  |  |  |
| --- | --- | --- |
| Идентификатор | Тип | Назначение |
| txtInputSum | Edit | Текстопое поле для ввода суммы денег |
| lstCoffeeKind | ListBox | Список с доступными напитками |
| lstOutput | ListBox | Окно выдачи (кофе и сдача) |
| btnCook | Button | Кнопка приготовления |
| btnClose | Button | Кнопка завершения работы |

# Экспериментальная часть

## Описание подхода к тестированию программы

Для тестирования составляются наборы файлов с исходными данными для инициализации приложения. Программа при запуске считывает количество имеющихся в наличии ресурсов с двух файлов:

* coffee.cfg содержит запасы кофейных напитков;
* cash.cfg содержит запасы наличности для сдачи.

Перед проведением тестирования необходимо сохранить исходные файлы coffee.cfg и cash.cfg, содержащие достаточные запасы ресурсов по умолчанию, в другую папку.

Перед проведением очередного теста, соответствующий набор файлов копируется под этими именами в папку с программой. После этого запускаем программу, проводим тестирование, закрываем программу, записываем результаты, восстанавливаем исходный набор файлов и переходим к следующему тесту.

В ходе тестирования необходимо проверить не только стандартное функционирование программы, но и его переход на граничные условия. В данном случае граничными условиями являются ограниченные запасы наличности для сдачи и ассортимент напитков – кофе и деньги могут закончиться и программа должна корректно обработать эти ситуации.

Для быстрой проверки каждой ситуации подготовим исходные наборы файлов с заведомо малым количеством соответствующего ресурса. Для проверки ситуации с окончанием кофейных порций будет использован файл coffee.0.cfg, содержимое которого приведено в таблице2.

Таблица 2. Содержимое исходного файла coffee.0.cfg

|  |  |  |
| --- | --- | --- |
| Название напитка | Цена за порцию | Количество порций |
| Cappuccino | 50 | 2 |
| Espresso | 25 | 1 |
| Americano | 30 | 1 |

Для проверки ситуации с окончанием наличности используем файл cash.0.cfg, содержимое которого приведено в таблице 3.

Таблица 3. Содержимое исходного файла cash.0.cfg

|  |  |
| --- | --- |
| Номинал банкноты/монеты | Количество |
| 500 | 2 |
| 100 | 2 |
| 50 | 2 |
| 10 | 2 |
| 5 | 2 |
| 2 | 2 |
| 1 | 2 |

## Тестирование

На базе принятого подхода составим следующий набор тестов:

1. ***Проверка ситуации закончившихся напитков (coffee.0.cfg).*** 
   1. Вносим 100 руб., готовим «Американо», нажимаем «Приготовить».

Программа должна выдать напиток стоимостью 30 рублей и сдачу – 70 рублей с минимально возможным в данном случае набором банкнот и монет (т.е. в виде одной 50-рублёвой банкноты и двух 10-рублёвых монет). Результат выполнения программы для этого тестового случая можно увидеть на рисунке 3.

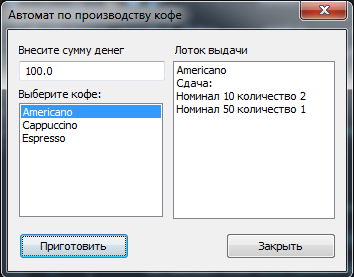


Рисунок 3. Пример нормального выполнения программы (сдача со 100 руб.).

* 1. Вносим 100 руб., готовим «Американо», нажимаем «Приготовить».

Поскольку мы загрузили всего одну порцию «Американо» и она была выдана на предыдущем шаге, автомат должен выдать сообщения об ошибке с пояснением, что такой кофе отсутствует в наличии. Вид главного окна с сообщением об ошибке изображён на рисунке 4.

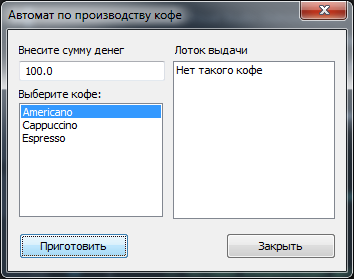


Рисунок 4. Пример сообщения об ошибке (напитка нет в наличии).

* 1. Вносим 1000 руб., готовим «Каппучино», нажимаем «Приготовить».

При инициализации программы загружено 2 порции этого напитка, так что мы видим нормальное выполнение программы. Штатный размен суммы в 100 рублей изображён на рисунке 5.

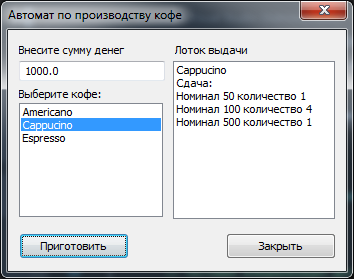


Рисунок 5. Пример нормального выполнения программы (сдача с 1000 руб.).

* 1. Вносим 100 руб., готовим «Каппучино», нажимаем «Приготовить».

В системе должна оставаться ещё одна порция «Каппучино», поэтому мы видим результат штатной выдачи напитка с разменом, подобный изображённому на рисунке 3.

* 1. Вносим 100 руб., готовим «Каппучино», нажимаем «Приготовить».

Поскольку последнюю порцию этого кофе мы забрали на предыдущем шаге, сейчас мы увидим сообщение об ошибке, как на рисунке 4.

* 1. Вносим 100 руб., готовим «Эспрессо», нажимаем «Приготовить».

Программа выполняется штатно – забираем последнюю и единственную порцию «Эспрессо» стоимостью 25 руб., автомат выдаёт заказанный напиток и сдачу со 100 рублей, как изображено на рисунке 6.

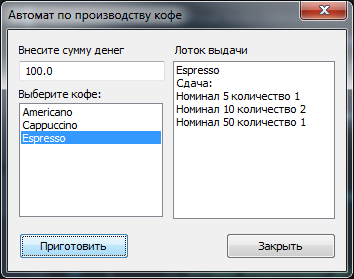


Рисунок 6. Пример нормального выполнения программы («Эспрессо»).

* 1. Вносим 100 руб., готовим «Эспрессо» , нажимаем «Приготовить».

Система ожидаемо уведомляет о закончившихся напитках, подобно тому, как показано на рисунке 4.

1. ***Проверка ситуации закончившейся сдачи (cash.0.cfg).***
   1. Вносим 1000 руб., готовим «Американо».

Сразу вносим большую сумму, чтобы поверить, что автомат с первого раза обработает эту ситуации. Появившееся сообщение об ошибке изображено на рисунке 7.

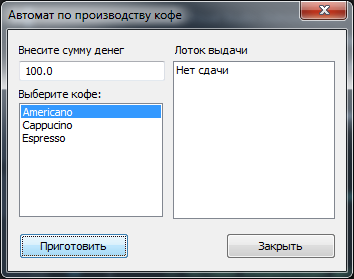


Рисунок 7. Пример сообщения об ошибке (в автомате нет сдачи).

1. ***Проверка ситуации закончившейся сдачи (cash.0.cfg).***
   1. Вносим 100 руб., готовим «Американо».

В этом тесте проверим для начала штатную работу программы, когда в автомате достаточно напитков для выдачи и наличности для размена. Результат приведён на рисунке 3.

* 1. Вносим 100 руб., готовим «Американо».

На предыдущем шаге должны были закончиться 10-рублёвые монеты, так что на этом шаге получаем сообщение о закончившейся наличности для сдачи, как изображено на рисунке 7.

По результату тестирования можно сделать вывод, что программа успешно прошла весь набор тестов, выдав ожидаемые результаты на всех наборах входных данных, включая критические.

## Инструкция пользователя

Перед запуском программы необходимо убедиться в наличии файлов конфигурации – coffee.cfg, в котором указаны начальные запасы кофейных напитков, и cash.cfg, в котором хранятся данные об имеющихся в начале работы банкнотах и монетах разного достоинства. Если этих файлов нет в папке с исполняемым файлом программы, эти запасы будут проинициализированы значениями по умолчанию. Запасы кофе в этом случае буду такими, как указано в таблице 4.

Таблица 4. Запасы кофе, задаваемые по умолчанию.

|  |  |  |
| --- | --- | --- |
| Название напитка | Цена за порцию | Количество порций |
| Cappuccino | 50 | 10 |
| Espresso | 25 | 10 |
| Americano | 30 | 10 |

В таблице 5 приведены запасы наличности, принимаемые в качестве настроек по умолчанию.

Таблица 5. Запасы наличных денег на сдачу, задаваемые по умолчанию.

|  |  |
| --- | --- |
| Номинал банкноты/монеты | Количество |
| 500 | 1 |
| 100 | 10 |
| 50 | 20 |
| 10 | 50 |
| 5 | 100 |
| 2 | 100 |
| 1 | 200 |

Работа с виртуальным кофейным автоматом начинается с запуска программы. Основное окно программы после запуска показано на рисунке 8.

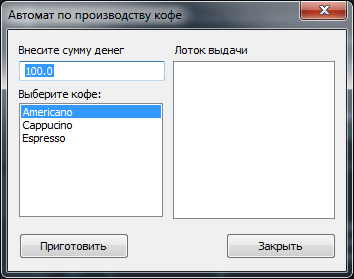


Рисунок 8. Главное окно программы после запуска.

Ассортимент кофейных напитков может отличаться в зависимости от того, что указано в конфигурационном файле coffee.cfg.

Входной информацией для автомата по приготовлению кофе являются сумма денег, вносимых в качестве предоплаты, и вид кофейного напитка.

Сумма денег вводится в соответствующем текстовом поле в левой верхней части главного окна. По умолчанию всегда задана сумма 100 рублей, её можно изменить.

Ниже поля для ввода предоплаты находится список кофейных напитков, которые были загружены в автомат в начале работы. После запуска программы всегда выбран первый пункт этого списка, впоследствии его можно изменять.

В правой части программы находится список результатов, представляющий собой виртуальный лоток выдачи кофе и сдачи.

Для приготовления кофе необходимо нажать соответствующую кнопку, находящуюся сразу под списком результатов. Перед приготовлением кофе, программа сперва проверяет остающиеся у неё запасы порций выбранного напитка. В случае, если в запасе нет ни одной порции, программа выдаст сообщение о том, что такого напитка нет, и завершит процедуру приготовления. Пользователь сможет продолжить работу, выбрав другой напиток.

Если же хотя бы одна порция имеется в наличии, программа следующим шагом проверяет, сможет ли она выдать сдачу, с учётов внесённой суммы и стоимости выбранного кофе. Если в кофейном автомате хранится достаточное количество наличных денег подходящего номинала, в виртуальном лотке выдачи появится заказанный напиток и сдача, с перечислением номиналов и количества купюр и монет, как показано на рисунке 9.

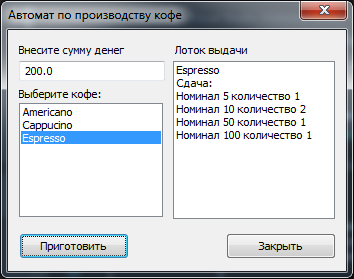


Рисунок 9. Пример заказа «Каппучино» с предоплатой в 200 руб.

В противном случае, в списке результатов появится сообщение об ошибке – автомат условно возвращает предоплату и отказывается готовить выбранный напиток. Пользователь может повторить попытку, введя другую сумму.

Таких попыток может быть сколь угодно много, и каждым успешным приготовлением напитка и выдачей сдачи происходит уменьшение хранящихся в автомате ресурсов, что в конечном итоге приведёт к регулярному появлению отказов с сообщением об ошибке.

Закрыв программу и запустив её заново, пользователь начинает сеанс работы с кофейным автоматом заново, имея в наличии полный запас ресурсов, указанный в двух конфигурационных файлах

# Заключение

В результате работы была спроектирована и разработана программная система, моделирующая функционирование автомата по приготовлению кофе. В ходе работы над проектом изучено множество аспектов проектирования сложных систем, включая использование языка UML и объектно-ориентированного подхода. Получены практические навыки разработки программ на объектно-ориентированном языке высокого уровня C++. Изучены особенности создания с помощью этого языка программирования и среды разработки Microsoft Visual Studio графического интерфейса пользователя.

Разработанная система является скорее демонстрационным приложением, однако содержит потенциал развития в программно-аппаратный комплекс, управляющий кофейным автоматом из реального мира. Если определить сущность «Кофе» как абстрактный базовый класс с виртуальным методом «Приготовить», а каждый вид кофейного напитка реализовать как отдельный производный класс, то в реализации этого метод можно вставить низкоуровневые системные вызовы ОС для управления периферийным оборудованием для приготовления реального напитка. Кроме того, такая иерархия позволит добавлять новые типы динамически, с помощью подключаемых модулей – динамически загружаемых библиотек.

# Список литературы

1. Боггс, М. UML и Rational Rose / М. Боггс. - Москва:РГГУ, 2010. -385 c.

2. Бородакий, Ю. В. Эволюция информационных систем / Ю.В. Бородакий, Ю.Г. Лободинский. - Москва:СИНТЕГ, 2011. - 368 c.

3. Буч, Гради Введение в UML от создателей языка / Гради Буч , Джеймс Рамбо ,Ивар Якобсон. - М.: ДМК Пресс, 2015. - 496 c.

4. Буч, Грейди Язык UML. Руководство пользователя / Гради Буч , Джеймс Рамбо ,Айвар Джекобсон. - М.: ДМК, 2015. - 432 c.

5. Роберт, А. Максимчук UML для простых смертных / Роберт А. Максимчук, Эрик Дж. Нейбург. - Москва: СИНТЕГ, 2014. - 272 c.

6. Пайлон, Д. UML 2 для программистов / Д. Пайлон. - М.: Питер, 2012. -198 c.

7. Фаулер, Мартин UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Мартин Фаулер. - Москва: СИНТЕГ, 2011. - 192 c.

8. Фельдман, Я. А. Создаем информационные системы (+ CD-ROM) / Я.А. Фельдман. - М.: Солон-Пресс, 2007. - 120 c. 21. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма и др. - Москва: СИНТЕГ, 2016. - 366 c.

9. Ларман, Крэг Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку / Крэг Ларман. - М.: Вильямс, 2013. - 736 c.

10. Йордон, Эдвард Объектно-ориентированный анализ и проектирование систем / Эдвард Йордон , Карл Аргила. - М.: ЛОРИ, 2014. - 264 c.

11. Гультяев, А. К. Проектирование и дизайн пользовательского интерфейса / А.К. Гультяев, В.А. Машин. - М.: Корона-Принт, 2010. - 350 c.

12. Грекул, В. И. Управление внедрением информационных систем / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. - Москва: РГГУ, 2014. - 224 c.

13. Н. Культин Основы программирования в Microsoft Visual C++ 2010 / Н. Культин. - М.: БХВ-Петербург, 2010. - 384 c.

# Приложения

Coffee.h

#pragma once

// Порция кофе

class Coffee

{

public:

// класс исключения (недостаточно порций)

class NotEnough {

public:

NotEnough(int count) : count(count) {

}

int getCount() const {

return count;

}

private:

int count;

};

Coffee(const tstring& kind = "", double price = 0.0, int count = 0);

inline tstring getKind() const { return kind; }

inline void setKind(const tstring& value) { kind = value; }

inline double getPrice() const { return price; }

inline void setPrice(double value) { price = value; }

inline int getCount() const { return count; }

inline void setCount(int value) { count = value; }

// забрать несколько порций (возвращает оставшееся количество)

int takeSome(int count);

// добавить несколько порций (возвращает оставшееся количество)

int addSome(int count);

private:

// тип кофе

tstring kind;

// цена порции

double price;

// количество порций

int count;

};

Coffee.cpp

#include "stdafx.h"

#include "Coffee.h"

Coffee::Coffee(const tstring& kind, double price, int count) : kind(kind), price(price), count(count)

{

}

int Coffee::takeSome(int count)

{

if (this->count < count)

throw NotEnough(this->count);

this->count -= count;

return this->count;

}

int Coffee::addSome(int count)

{

this->count += count;

return this->count;

}

CoffeeMashine.h

#pragma once

class Coffee;

// Номинал купюры или монеты

enum CashValue {

coin1 = 1,

coin2 = 2,

coin5 = 5,

coin10 = 10,

note50 = 50,

note100 = 100,

note500 = 500,

};

// Кофейный автомат

class CoffeeMashine

{

public:

typedef std::map<CashValue, int> Cash;

typedef std::map<tstring, Coffee> CoffeeAvail;

// Классы исключений:

// Базовый класс всех исключений

class Error {

public:

Error(const tstring& kind) : kind(kind) {

}

private:

// тип кофе

tstring kind;

};

// Порций данного типа нет в наличии

class NotAvail : public Error {

public:

NotAvail(const tstring& kind) : Error(kind) {

}

};

// На выбранную порцию не хватает внесённой суммы

class NotEnoughMoney : public Error {

public:

NotEnoughMoney(const tstring& kind, double diff) : Error(kind), diff(diff) {

}

// сколько не хватает

double diff;

};

// Нет такого набора купюр, чтобы выдать сдачу с внес1нной суммы

class NoChange : public Error {

public:

NoChange(const tstring& kind, double remainder)

: Error(kind), remainder(remainder) {

}

private:

// сколько не удалось разменять

double remainder;

};

// Конструктор

CoffeeMashine(const tstring& coffeeFileName = "", const tstring& cashFileName = "");

// Просмотр ассортимента с ценником:

// возвращает перечень имеющеегося в наличии кофе

const CoffeeAvail& getCoffeeAvail() const {

return coffeeAvail;

}

// Приготовление кофе:

// принимает тип кофе и сумму денег

// возвращает порцию кофе и сдачу

std::pair<Coffee, CoffeeMashine::Cash> Cook(const tstring& kind, double sum);

private:

// доступные порции кофе

CoffeeAvail coffeeAvail;

// доступная наличность (пары номинал-количество)

Cash cashAvail;

};

CoffeeMashine.cpp

#include "stdafx.h"

#include "CoffeeMashine.h"

#include "Coffee.h"

CoffeeMashine::CoffeeMashine(const tstring& coffeeFileName, const tstring& cashFileName)

{

tstring coffeeKind;

if (coffeeFileName.empty()) {

// тестовая инициализация доступных порций кофе

coffeeKind = "Капучино";

coffeeAvail.insert(CoffeeAvail::value\_type(

coffeeKind, Coffee(coffeeKind, 50.0, 10)

));

coffeeKind = "Эспрессо";

coffeeAvail.insert(CoffeeAvail::value\_type(

coffeeKind, Coffee(coffeeKind, 25.0, 10)

));

coffeeKind = "Американо";

coffeeAvail.insert(CoffeeAvail::value\_type(

coffeeKind, Coffee(coffeeKind, 30.0, 10)

));

} else {

std::ifstream input(coffeeFileName);

if (input) {

int count;

double price;

while (input >> coffeeKind >> price >> count) {

for (int i = 0; i < count; i++) {

coffeeAvail.insert(CoffeeAvail::value\_type(

coffeeKind, Coffee(coffeeKind, price, count)

));

}

}

}

}

if (cashFileName.empty()) {

// тестовая инициализация доступных банкнот и монет

cashAvail[note500] = 1;

cashAvail[note100] = 10;

cashAvail[note50] = 20;

cashAvail[coin10] = 50;

cashAvail[coin5] = 100;

cashAvail[coin2] = 100;

cashAvail[coin1] = 200;

} else {

std::ifstream input(cashFileName);

if (input) {

int value, count;

while (input >> value >> count) {

cashAvail[static\_cast<CashValue>(value)] = count;

}

}

}

}

std::pair<Coffee, CoffeeMashine::Cash> CoffeeMashine::Cook(const tstring& kind, double sum)

{

// Поиск имеющейся в наличии порции

auto coffee = coffeeAvail.find(kind);

// Порций такого кофе нет?

if (coffee == coffeeAvail.end())

throw NotAvail(kind);

// Расчёт сдачи

double change = sum - coffee->second.getPrice();

if(change < 0)

throw NotEnoughMoney(kind, -change);

// Набираем сдачу из имеющихся купюр и монет

Cash cash;

auto it = cashAvail.rbegin();

// Перебираем с конца (map отсортирован по ключу, так что в конце крупные)

while (it != cashAvail.rend()) {

bool ranOut = false;

while (change >= it->first) {

change -= it->first;

cash[it->first]++;

it->second--;

if (!it->second) {

it = static\_cast<decltype(it)>(

cashAvail.erase(std::next(it).base())

);

ranOut = true;

if (it == cashAvail.rend()) {

break;

}

}

}

if (change == 0) {

break;

}

if (!ranOut) {

it++;

}

}

if (change != 0)

throw NoChange(kind, change);

// забираем одну порцию кофе из запасов

std::pair<Coffee, Cash> result(coffee->second, cash);

if(0 == coffee->second.takeSome(1)) {

// порции этого кофе закончились

coffeeAvail.erase(coffee);

}

return result;

}

main.cpp

#include "stdafx.h"

#include "Resource.h"

#include "CoffeeMashine.h"

#include "Coffee.h"

// Автомат по производству кофе

CoffeeMashine\* coffeeMashine = nullptr;

HWND hlstCoffee = nullptr;

HWND hlstOutput = nullptr;

// Главная оконная процедура (обработчик сообщений главного окна приложения)

INT\_PTR CALLBACK DialogProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

TCHAR buffer[256];

UINT ctrlID = 0;

switch(uMsg) {

// Сообщение от одного из элементов управления, размещённых на диалоговом окне

case WM\_COMMAND:

// Идентификатор элемента управления, являющегося источниом сообщения

ctrlID = LOWORD(wParam);

switch (ctrlID) {

case IDCANCEL: // Нажатие ESC

SetDlgItemText(hDlg, txtInputSum, \_T("0.0"));

return TRUE;

case btnClose: // Нажатие кнопки "Закрыть"

SendMessage(hDlg, WM\_CLOSE, 0, 0);

return TRUE;

case btnCook: // Нажатие кнопки "Приготовить"

// очистить список результатов

SendMessage(hlstOutput, LB\_RESETCONTENT, (WPARAM)0, (LPARAM)0);

try {

// прочитать денежную сумму

GetDlgItemText(hDlg, txtInputSum, buffer,

sizeof(buffer) / sizeof(buffer[0]) - 1);

tistringstream is(buffer);

double sum = 0;

is >> sum;

// прочитать выбранный вид кофе

TCHAR text[256];

int index = SendMessage(hlstCoffee, LB\_GETCURSEL,

(WPARAM)0, (LPARAM)0);

SendMessage(hlstCoffee, LB\_GETTEXT, (WPARAM)index, (LPARAM)text);

// приготовить кофе и получить сдачу

std::pair<Coffee, CoffeeMashine::Cash> result =

coffeeMashine->Cook(text, sum);

// вывод результатов

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0, (LPARAM)text);

if (result.second.empty()) {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Сдача не требуется"));

} else {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Сдача:"));

auto it = result.second.begin();

for (; it != result.second.end(); it++) {

tostringstream os;

os << \_T("Номинал ") << it->first

<< \_T(" количество ") << it->second;

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)os.str().c\_str());

}

}

}

catch (CoffeeMashine::NotAvail& e) {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Нет такого кофе"));

}

catch (CoffeeMashine::NotEnoughMoney& e) {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Недостаточно денег"));

}

catch (CoffeeMashine::NoChange& e) {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Нет сдачи"));

}

catch (CoffeeMashine::Error& e) {

SendMessage(hlstOutput, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)\_T("Не удалось приготовить кофе"));

}

return TRUE;

}

break;

case WM\_INITDIALOG:

// В обработчике этого сообщения удобно проводить

// начальную настройку приложения

hlstCoffee = GetDlgItem(hDlg, lstCoffeeKind);

hlstOutput = GetDlgItem(hDlg, lstOutput);

// Выделяем память для объекта "Автомат по производству кофе"

coffeeMashine = new CoffeeMashine("coffee.cfg", "cash.cfg");

SetDlgItemText(hDlg, txtInputSum, \_T("100.0"));

{

CoffeeMashine::CoffeeAvail coffeeAvail =

coffeeMashine->getCoffeeAvail();

// Заполняем список выбора напитков

for (const auto coffee : coffeeAvail) {

SendMessage(hlstCoffee, LB\_ADDSTRING, (WPARAM)0,

(LPARAM)coffee.first.c\_str());

}

}

SendMessage(hlstCoffee, LB\_SETCURSEL, (WPARAM)0, 0);

break;

case WM\_CLOSE:

/\* Это сообщение вызывается перед закрытием окна. Здесь можно спросить

пользователя, действительно ли он хочет завершить работу (особенно при

наличии несохранённых данных). В этом приложении мы просто отдаём команду

на закрытие главного окна. \*/

DestroyWindow(hDlg);

return TRUE;

case WM\_DESTROY:

/\* Мы должны обработать это сообщение, чтобы сказать системе, что

хотим завершить основной поток приложения. Для этого служит функция

PostQuitMessage(). Сейчас есть последняя возможность для освобождения

выделенных ресурсов. \*/

delete coffeeMashine;

PostQuitMessage(0);

return TRUE;

}

return FALSE;

}

// Точка входа приложения

int WINAPI \_tWinMain(HINSTANCE hInst, HINSTANCE h0, LPTSTR lpCmdLine, int nCmdShow)

{

MSG msg;

BOOL ret;

// Создаём главное окно

HWND hDlg = CreateDialogParam(hInst, MAKEINTRESOURCE(mainWnd), 0, DialogProc, 0);

// Отображаем его на экране

ShowWindow(hDlg, nCmdShow);

// Стандартным образом обрабатываем поступающие сообщения в цикле

while((ret = GetMessage(&msg, 0, 0, 0)) != 0) {

if(ret == -1)

return -1;

if(!IsDialogMessage(hDlg, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return 0;

}