Федеральное агентство связи

Уральский технический институт связи и информатики (филиал)

ФГБОУ ВО «Сибирский государственный университет телекоммуникаций и информатики» в г. Екатеринбурге

(УрТИСИ СибГУТИ)

Утверждаю

Зав. Кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Информационных систем и технологий

Разработка графического приложения для составления недельного рациона питания

Курсовой проект

09.03.01.000056 П.956 КП

Выполнил

студент (ка) гр. Такшеев К.А.

ФИО подпись

Руководитель Кислицин Е.В.

ФИО подпись

г. Екатеринбург. 2020г.

Отзыв руководителя

Федеральное агентство связи

Уральский технический институт связи и информатики (филиал)

ФГБОУ ВО «Сибирский государственный университет телекоммуникаций и информатики» в г. Екатеринбурге

(УрТИСИ СибГУТИ)

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовой проект

по дисциплине: «Объектно-ориентрованное программирование»

на тему: «Разработка графического приложения для составления недельного рациона питания»

студента(ки) группы ПЕ-71б ФИО Такшеев К.А.

ИСХОДНЫЕ ДАННЫЕ

Разработать программное обеспечение с графическим интерфейсом на предпочтительном языке программирования с использованием объектно-ориентированного подхода:

* с использование специальных средств языка;
* с использованием паттернов программирования;
* с использованием принципов объектно-ориентированного подхода.

Содержание

[Введение 5](#_Toc59587772)

[1 Анализ и описание предметной области 6](#_Toc59587773)

[1.1 Предметная область 6](#_Toc59587774)

[1.2 Словарь здорового питания 7](#_Toc59587775)

[1.3 Анализ существующих программных решений 8](#_Toc59587776)

[1.3.1 «Калькулятор рациона питания» (АСПОН - Питание) 8](#_Toc59587777)

[1.3.2 "Калькулятор калорий" 10](#_Toc59587778)

[2 Разработка программного обеспечения 12](#_Toc59587779)

[2.1.1 Планирование объектов 12](#_Toc59587780)

[2.1.2 Программная реализация спланированных классов 13](#_Toc59587781)

[2.1.2.1 Класс продуктов 13](#_Toc59587782)

[2.1.2.2 Класс правил 14](#_Toc59587783)

[2.1.2.3 Класс базы данных 15](#_Toc59587784)

[2.1.2.4 Класс блюд 17](#_Toc59587785)

[2.1.2.5 Класс дней недели 17](#_Toc59587786)

[2.1.2.6 Класс недели и использование паттерна “Proxy” 17](#_Toc59587787)

[2.1.2.7 Класс генератора расписания и паттерн “Абстрактный метод” 19](#_Toc59587788)

[2.1.2.7.1 Алгоритм составления блюда 20](#_Toc59587789)

[2.1.2.7.2 Алгоритм составления дневного рациона 22](#_Toc59587790)

[2.1.2.7.3 Алгоритм составления недельного рациона 22](#_Toc59587791)

[2.2 Проектирование пользовательского интерфейса 23](#_Toc59587792)

[2.2.1 Графический интерфейс главного меню 23](#_Toc59587793)

[2.2.2 Графический интерфейс недели 24](#_Toc59587794)

[2.2.3 Графический интерфейс рациона питания на день 25](#_Toc59587795)

[2.3 Тестирование 26](#_Toc59587796)

[2.4 Плюсы и минусы разработанного программного обеспечения. 26](#_Toc59587797)

[2.5 Руководство пользователя 27](#_Toc59587798)

[Заключение 28](#_Toc59587799)

[Список использованных источников 29](#_Toc59587800)

[Приложение А 30](#_Toc59587801)

Введение

Компьютерные технологии развиваются все больше и больше с каждым днем, и уметь пользоваться компьютером, в наше время, по большей части определяет степень квалифицированности специалиста. Разработчики программ ставят для себя каждый раз все большиецели. Сначала это было, уменьшение занимаемой памяти программой из-за не сильно мощного компьютерного оборудования, сейчас же, с нынешней компьютерной техникой, задача состоит в том, чтобы как можно сильнее сократить время разработки и исполнения программой своей задачи. А также простота чтения написанного кода, так как, в нынешнее время, очень редко, одну программу пишет один человек. Разработкапрограмм происходит в команде. А для того, чтобы каждый разработчик мог разобраться в написанном ранее коде другим разработчиком, нужно упрощать написание программы, разбивая ее на блоки.

Для достижения этих целей в отрасли создания программных комплексов используют методы и подходы управления процессом разработки.

На разных этапах развития программной инженерии использовались различные технологии программирования:

• императивное программирование;

• модульное программирование;

• структурное программирование;

• программирование, управляемое данными;

• программирование, управляемое событиями;

• функциональное программирование;

• логическое программирование и т.п.

Теперь невозможно принять участие в дискуссии, посвященной программированию, если не использовать термин "объектно-ориентированное программирование".

Целькурсовой работы: закрепить и углубить навыки в использовании объектно-ориентированного подхода к программированию, полученные в процессе изучения дисциплины «Основы и объектно-ориентированного программирования».

Задачи курсовой работы: для выполнения поставленной задачи, предлагается использовать средство объектно-ориентированного языка C#. Предполагается разработать консольное приложение, которое осуществит определенную сортировку, для упорядочения заданных данных по условию задачи, которая будет производить сортировка по одному из заданных ключей, в итоге упорядочив всю полученную информацию.

1 Анализ и описание предметной области

1.1 Предметная область

Правильное питание – залог хорошего самочувствия, здорового внешнего вида и крепкого иммунитета. Сбалансированное здоровое питание позволяет получать все необходимые минералы и микроэлементы. Если же стоит задача снизить вес, соблюдение принципов правильного питания позволит сделать это без дискомфорта и голода.

Суточная калорийность рассчитывается индивидуально, исходя из веса, возраста, пола и активности. Например, для сотрудницы офиса 25-30 лет среднего телосложения, которая ходит в день менее 30 минут, для поддержания веса достаточно будет 1600-1700 ккал, а для её ровесника, который 3 раза в неделю интенсивно тренируется, оптимальным будет дневной рацион калорийностью около 2500 ккал.

Для здорового снижения веса дневную калорийность достаточно снизить на 15-20% - этого вполне достаточно, чтобы постепенно терять лишние килограммы. Часто к правильному питанию относят диеты, которые предполагают снижение дневной калорийности до 1000-1200 калорий, резкое сокращение количества жиров или углеводов. Это большая ошибка! Полуголодный рацион не может быть здоровым.

Резкое снижение дневной нормы калорий заставляет организм работать в авральном режиме, сигнализирует о том, что еда кончилась, и, чтобы выжить, нужно экономить ресурсы. Организм при этом переходит в режим выживания – замедляет обмен веществ, сокращает расход калорий на питание мозга, а также старается запасти побольше жира. После такой диеты велик шанс набрать куда больше веса, чем было сброшено. Кроме того, голодание может нарушить важные процессы в организме и даже оказаться смертельно опасным.

Вода – это основной элемент правильного питания. Без воды организм не способен функционировать. Поэтому очень важно пить столько, сколько требует организм. Для поддержания оптимального уровня водного баланса нужно выпивать как минимум два литра в сутки. Полезнее всего пить чистую воду, а не чай кофе или сок. Делать это лучше отдельно от приема пищи, чтобы не разбавлять желудочный сок.

Если вы почувствовали голод, попробуйте сначала выпить воды. Иногда мозг ошибочно принимает сигналы о жажде за сигналы голода. Если чувство голода при этом прошло – значит, прием пищи можно отложить.

1.2 Словарь здорового питания

Веганы — исключают из своего рациона все продукты животного происхождения (мясо, рыба, птица, молочные продукты и иногда мёд).

Вегетарианцы — едят всё за исключением мяса, птицы, рыбы.

Калорийность (от калории — старой единицы измерения энергии): Энергетическая ценность пищи — количество энергии, получаемой организмом из продукта питания. Теплота сгорания — количество энергии, выделяющейся при полном сгорании вещества.

Белки (протеины, полипептиды) — высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью. В живых организмах аминокислотный состав белков определяется генетическим кодом, при синтезе в большинстве случаев используется 20 стандартных аминокислот. Множество их комбинаций создают молекулы белков с большим разнообразием свойств. Кроме того, аминокислотные остатки в составе белка часто подвергаются посттрансляционным модификациям, которые могут возникать и до того, как белок начинает выполнять свою функцию, и во время его «работы» в клетке. Часто в живых организмах несколько молекул разных белков образуют сложные комплексы, например фотосинтетический комплекс.

Жиры, также триглицериды, триацилглицериды (сокр. ТАГ) — органические вещества, продукты этерификации карбоновых кислот и трёхатомного спирта глицерина. В живых организмах выполняют, прежде всего, структурную и энергетическую функции: они являются основным компонентом клеточной мембраны, а в жировых клетках сохраняется энергетический запас организма.

Углеводы являются неотъемлемым компонентом клеток и тканей всех живых организмов представителей растительного и животного мира, составляя (по массе) основную часть органического вещества на Земле. Источником углеводов для всех живых организмов является процесс фотосинтеза, осуществляемый растениями.

Здоровое питание — это питание, обеспечивающее рост, нормальное развитие и жизнедеятельность человека, способствующее укреплению его здоровья и профилактике заболеваний. Соблюдение правил здорового питания в сочетании с регулярными физическими упражнениями сокращает риск хронических заболеваний и расстройств, таких как ожирение, сердечно-сосудистые заболевания, диабет, повышенное давление и рак. Современная наука проводит многочисленные исследования, чтобы оптимизировать рацион питания для профилактики основных хронических заболеваний.

1.3 Анализ существующих программных решений

В современном мире эта ниша очень развита и имеет множество программных и аналитических решений.

На просторах интернета можно найти множество фитнес программ считалок калорий. Как правило, это простые в исполнении программки, считающие калорийность рациона и содержание основных нутриентов (белков, жиров и углеводов), но есть отдельные программы, которые подходят к вопросу питания более детально.

1.3.1 «Калькулятор рациона питания» (АСПОН - Питание)

Пакет программных средств «Калькулятор рациона питания» (АСПОН - Питание) Предназначен для анализа рациона питания на предмет качественной и количественной оценки его сбалансированности и для выработки рекомендуемого сбалансированного рациона питания. С его помощью могут быть составлены сбалансированные индивидуальные и коллективные рационы питания в семье, в воспитательных и образовательных учреждениях, в учреждениях общественного питания и др.

Пакет программных средств (ППС) также может быть использован во врачебной практике, в том числе в комплексе Аспон-Д, с целью профилактического обследования и диагностики. Пользователями могут быть врач-диетолог, практикующий врач, а также другие лица, имеющие отношение к технологии питания, обученные работе с программой и получающие советы врачадиетолога. Субъектом программы является Пациент, чей конкретный рацион анализируется и корректируется с помощью программы. Пациентом может быть отдельный человек или целый коллектив. В тех случаях, когда требуется оценить качество кулинарных изделий или меню школьного, спортивного, ресторанного т.д. или когда для них подбирается оптимальный состав продуктов, с ними надо работать как с пациентами. Программные возможности: анализ представленного существующего рациона пациента на содержание нутриентов; выбор для пациента норм потребления нутриентов (Стандарта).

Сравнение рациона со Стандартом и выявление недостаточности или избыточности потребления требуемых по Стандарту нутриентов; корректировка представленного рациона с выработкой рекомендуемого рациона, отвечающего Стандарту. Программа позволяет формировать индивидуальные рационы, рационы для детских, спортивных и др. коллективов, для учебных и воспитательных учреждений, для учреждений общественного питания, в практике семейного питания и т.п. Важным назначением программы является формирование рационов для различных диет.

Вс определяется тем, какие в программе используются стандарты. Сильными сторонами программы являются: глубокий научный подход к подбору рациона питания, учет дефицита\избытка витаминов, микроэлементов и минералов, поступающих с пищей. Возможность применения программы на различном контингенте пользователей. Большая база данных по продуктам питания. Минусами программы являются: Необходимость работы диетолога с программой. Отсутствие учета двигательной активности и соотношения с потребляемыми калориями. Отсутствие адаптации к использованию спортсменами, нет четкой специализации.

1.3.2 "Калькулятор калорий"

Программа "Калькулятор калорий" Это программа для расчета калорийности еды и расхода потраченных калорий. Программа вычисляет баланс калорийности, сопоставляя потребленные с пищей калории и калории, израсходованные при физической активности. Программа основывается на ведении ежедневного хронологического дневника, куда пользователь заносит измерения объемов тела, вес, съеденные блюда и физические нагрузки. Вычисляется калорийность и пищевая ценность (белки, жиры, углеводы) потребленных продуктов, потраченные при физической активности калории, баланс калорийности за день.

Особенностью программы является то, что данные для просмотра могут быть отобраны за произвольный период; при этом автоматически рассчитываются итоговые и средние показатели за выбранный период - потребленные и потраченные калории в сумме, итоговый баланс калорий за период, баланс белков, жиров и углеводов. Динамика изменений ключевых показателей отображается на диаграммах. Также в программе имеется возможность составления, сохранения и загрузки в дневник рационов питания.

По каждому рациону рассчитывается его пищевая ценность и калорийность. В программе существует возможность добавления своих собственных продуктов или готовых блюд, удаления и редактирования блюд, а также возможность редактирования/добавления/удаления физических нагрузок. Сильными сторонами программы являются: программа имеет возможность производить подсчет энергозатрат по видам деятельности.

Имеет неплохую базу данных по продуктам питания и блюдам Минусами программы являются: база данных продуктов и блюд не учитывает содержания в них микро- и макроэлементов. Отсутствие суплиметов в базе данных. Нет четкой специализации программы. Отсутствие анализа рациона и выдачи рекомендаций. Программы выполнена на программной базе Microsoft Access, что подразумевает наличие у пользователя предустановленного расширенного пакета Microsoft Office. Программа работает только на 32х разрядных системах. Нет продуманного интерфейса и диалога с пользователем.

1.4 Техническое задание

Для разработки технического задания определим, что мы хотим по итогу получить.

Хочется получить компьютерное приложение с графическим интерфейсом.

Подойдем к разработке графического интерфейса.

Требуется что бы он был понятен, так же требуется что бы был простой доступ к каждому дню недели и к каждому приему пищи.

Экранные формы должны быть выполнены с использованием стандартных компонентов, предоставленных средой разработки.

Для самого же приложения требуется ввести ограничения. В первоначальной версии данного программного обеспечения предполагается ввод набора правил, который можно изменить, не нарушая всю целостность кода, то есть не требуется изменения больше одного класса, в данном случает класса правил (Rules).

Требуется составить так же расписание приемов пищи:

1. На один прием пищи
2. на один день
3. на одну неделю

Требуется ввести считывание баз данных существующих продуктов из файла или из удаленной базы данных.

Данные должны иметь собственные модели, от которых должны быть реализованы стандартные реализации.

Для достижения правильной архитектуры должны использоваться паттерны, структуры и принципы, стандартные для приложений использующие принцип объектно-ориентированного подхода.

2 Разработка программного обеспечения

2.1.1 Планирование объектов

Начиная разработку программного обеспечения с использованием объектно-ориентированного подхода стоит начать с выделения основных объектов (классов).

Основные объекты, которые мы будем использовать можем разделить на 2 вида:

1. Базовые;
2. Инкапсулирующие;
3. Объекты, использующие в себе базовые или инкапсулирующие объекты.

К базовым объектам можно отнести:

1. Структура для хранения продуктов;
2. Продукт;
3. Правила для алгоритма генерации рациона питания.

К инкапсулирующим отнесем:

1. Правила с дополнительными свойствами.

Объекты, использующие в себе базовые или инкапсулирующие объекты:

1. Блюдо;
2. День;
3. Неделя;
4. База данных;
5. Класс для создания расписания.

Класс продуктов содержит в себе такие характерные показатели как:

1. Содержание килокалорий;
2. Содержание жиров;
3. Содержание белков;
4. Содержание углеродов;
5. Масса.

Класс блюдо содержит набор продуктов.

Класс дня содержит набор блюд.

Класс недели содержит набор дней.

Правила с дополнительными свойствами содержит продукт, которые является ограничителем и контрольным продуктов, который определяет оптимальное содержание белков, жиров, углеводов и т.д.

Класс базы данных будет содержать универсальные методы для работы будь то удалённая или локальная (самодельная) база данных.

2.1.2 Программная реализация спланированных классов

2.1.2.1 Класс продуктов

Подойдем к реализации нашего программного продукта создав класс продуктов.

Создадим интерфейс IProduct, который будет содержать все выше перечисленные поля. Типы полей определяется характеристикой, которую они определяют. Так поле имя продукта будет типом строки (string), а все остальные поля будут иметь тип числа с плавающей запятой (double). Дополнительно так же расширим данный интерфейс интерфейсом ICloneable, который показывает, что можно создавать копию данного объекта и определим метод Clone с помощью которого и будет создавать клон объекта. Программная реализация интерфейса представлена на рисунке 2.1.2.1.1.

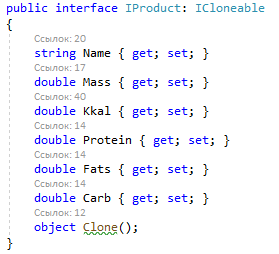


Рисунок 2.1.2.1.1 – Интерфейс продуктов

Создадим стандартную реализацию данного интерфейса. Для этого создадим класс DefaultProduct, унаследуем интерфейс IProduct, и реализуем все методы, описанные в данном интерфейсе, применяя к всем полям из интерфейса модификатор доступа publick.

Добавим так же конструктор 2 конструктора для данного класса. Один без параметров, а второй с определением всех полей, которые содержатся в классе. Реализацию данных конструкторов можно увидеть на рисунке 2.2.2.

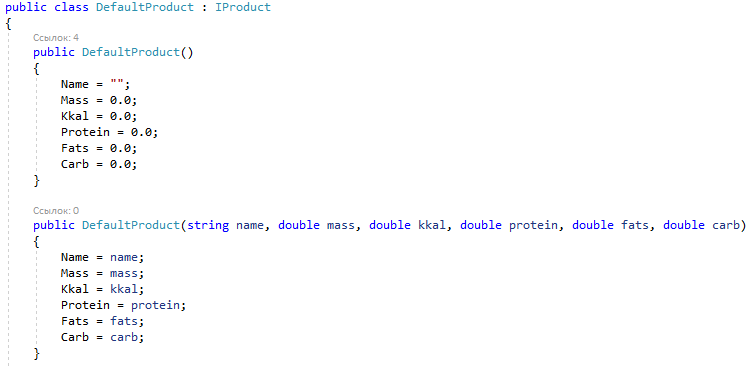


Рисунок 2.1.2.1.1 – Реализация конструкторов

2.1.2.2 Класс правил

Создадим следующий базовый класс – класс правил.

Создадим интерфейс IRules, который содержит одно лишь поле, это поле правил имеющее тип созданного ранее интерфейса IRules. Интерфейс представлен на рисунке 2.1.2.2.1.

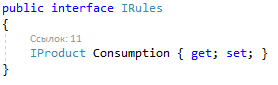


Рисунок 2.1.2.2.1 – Реализация интерфейса правил

Создадим стандартную реализацию данного интерфейса. Создадим класс DefaultRules и унаследуем интерфейс IRules. Создадим общедоступный конструктор, в котором определить наши стандартные правила, которые в последствии будет использоваться в создание рациона питания. Реализацию класса можно увидеть на рисунке 2.1.2.2.2.

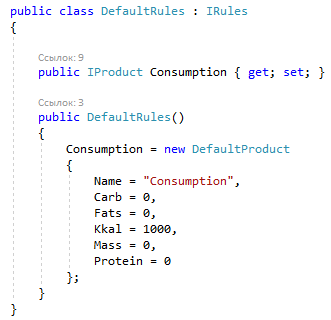


Рисунок 2.1.2.2.2 – Реализация стандартного класс для правил

2.1.2.3 Класс базы данных

Перейдем к реализации базы данных.

Создадим интерфейс IDB с тремя входными параметрами, которые обозначают типы данных которые будем хранить в нашей базе данных, а также разработаем интерфейс так, чтобы можно было легко подстроить под него любую базу данных.

Определим основные методы интерфейса.

Метод поиска данных по ключу (GET), который будет принимать на вход ключ и возвращать массив всех данных которые мы найдем в нашей базе данных.

Метод добавления данных по ключу (SET), который на вход будет принимать ключ и значение ключа, которые будут хранится по этому ключу.

Метод обновления данных по ключу (UPDATE), который будет на вход принимать ключ записи, которую надо изменить и новые данные для данного ключа.

Метод получения всех данных из базы данных (GETALL), который будет возвращать все данных из базы данных.

Реализацию данного интерфейса продемонстрирована на рисунке 2.1.2.3.1.

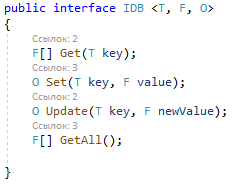


Рисунок 2.1.2.3.1 – Реализация интерфейса

Создадим абстрактный класс, в котором определим все входные типы данных для нашей стандартной реализации. Кроме того добавим некоторые поля, которые понадобятся для нашей стандартной реализации, это поле переменной в которой будем хранить нашу базу данных, обозначению методом доступа protected, потому что мы не хотим что бы класса которые наследовались от нашей стандартной реализации напрямую получали доступ к базе данных, таким образом мы абстрагируемся от реализации базы данных в классах наследниках. Так как у нас нет готовой базы данных, мы создадим свою и будем хранить ее в файле, для того что бы поместить ее в нашу программу нам понадобится метод для заполнения нашей базы данных. Определим же метод, который по имени файла, будет заполнять нашу базу данных.

Чтение из файла будем производить с помощью встроенного функционала языка C#. И считываемые строки будем преобразовывать в экземпляры класса продуктов, которые уже и будем хранить в нашей базе данных.

Реализацию данного абстрактного класса можно увидеть на рисунке 2.1.2.3.2.

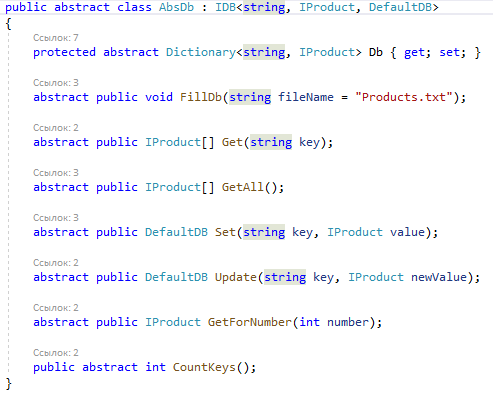


Рисунок 2.1.2.3.2 – Реализация абстрактного класс

Создадим стандартную реализацию наше базы данных. Как структуру данных будем использовать Словарь (Dictionary) с ключами в виде строк (string), а значениями данных ключом будет в виде продуктов, но введем абстракцию и укажем интерфейс продуктов IProduct, а все методы по их логическому значению реализуем в унаследованных методах.

2.1.2.4 Класс блюд

Создадим структуру для хранения блюда. Для этого создадим интерфейс IDish. В котором будут, поля для хранения имени блюда, а также будет поле в которой будем хранить все продукты, которые входят в блюдо, чтобы не привязывать данную структуру к определенной реализации продуктов, укажем как тип хранимых объектов интерфейс IProduct. И расширим нашу реализацию интерфейсом ICloneable и определим соответствующий метод Clone.

Создадим стандартную реализацию определив все поля, метод для генерации имению блюда определим, как строка, состоящая из первых двух букв имени всех продуктов, входящих в блюдо.

2.1.2.5 Класс дней недели

Структура для не отличается принципиальными изысками от класса блюд. Поэтому так же создадим интерфейс IDay, в котором определим поля, такие как, поле для хранения блюд, так как у нас интерфейс блюд является генерируем укажем передающийся тип данных в интерфейсе дня и укажем его в интерфейсе объектов нашего хранилища. И расширим наш интерфейс интерфейсом ICloneable, что бы была возможность клонировать объект.

Создадим стандартную реализацию в виде класса DefaultDay. Для это унаследуем класс от созданного ранее нами интерфейса, как параметр передав тип double. И реализовав все метода интерфейса.

2.1.2.6 Класс недели и использование паттерна “Proxy”

Как и в прошлые разы начнем создание нашего класса с универсального интерфейса IWeek, который будет принимать типы данных которые принимает интерфейс IDay, потому что одно из полей этого класса будет словарь из всех дней и советующие их дни с блюдами и продуктами. И сделаем возможность клонировать экземпляры классов, наследующие данный интерфейс расширив наш интерфейс интерфейсом ICloneable.

Но вот словарь мы будем использовать свой, потому что мы не хотим, чтобы в нашем словаре было больше 7 записей, потому что дней недели всего 7.

Для этого создадим класс MyDictionary и унаследуем его от универсального интерфейса словарей IDictionary и Dictionary, что бы мы могли в будущем его использовать и как интерфейса и как экземпляр стандартной реализации интерфейса словарей.

Прокси используется в том случае, если мы хоти сохранить старое поведение объекта и хотим добавить какое-либо поведение “сверху”, в нашем случае это будет проверка на количество записей в словаре при добавлении записи в словарь, если записей уже 7, как и дней в неделе, мы будем вызывать ошибку переполнения с пояснением ошибки. Так же на всякий случай укажем в комментариях к данному классу, чем отличается наш класс от стандартного. Дополнительный функционал поместим в метод Add, реализацию которого в нашем классе можно увидеть на рисунке 2.1.2.6.1.

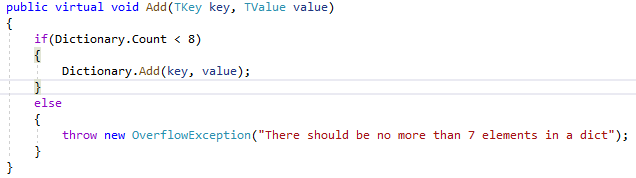


Рисунок 2.1.2.6.1 – Дополнительный функционал

Реализуем данный интерфейс в стандартном классе DefaultWeek унаследовав от интерфейса IWeek и предав в интерфейс тип данных double. Так же нашему полю дней недели присвоим стандартное значение в виде пустого нового словаря, который является экземпляром нашего класса словарей, но не будем как тип данных указывать наш словарь, что бы не привязываться к какой-либо конкретной реализации словаря, что бы при наследовании интерфейса другим классом мы могли поместить в переменную любую реализацию словарей, которая наследуют класс Dictionary, либо интерфейс IDictionary.

Так же создадим 2 конструктора. Один с пустыми параметрами, а второй с параметром в виде словаря, если у нас будет какой то уже готовый словарь, мы не заполняли его дополнительную команду, а сразу передавали в конструктор класса и присваивали его. Реализацию данного класса можно увидеть на рисунке 2.1.2.6.2.

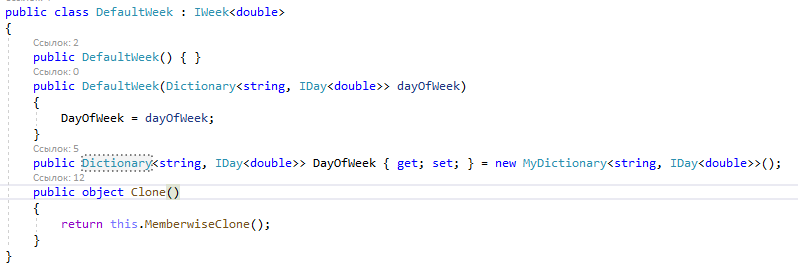


Рисунок 2.1.2.6.2 – Реализация

2.1.2.7 Класс генератора расписания и паттерн “Абстрактный метод”

Данный класс является одной из конечных точек курсовой работы. Именно в нем будет реализован основной алгоритм реализации нашего генератора расписания, а также именно в него будут инкапсулированы все созданные ранее нами классы и интерфейсы.

Создадим универсальный интерфейс, в котором будут лишь 3 метода:

1. Метод генерации блюда;
2. Метод генерации дневного рациона;
3. Метод генерации недельного рациона.

Так же сделаем добавим обобщающие параметры нашему интерфейсу, а именно:

1. Параметр, который является типом данных блюда;
2. Параметр, который является типом данных дня;
3. Параметр, который является типом данных дня;

Но мы не буем позволять передавать какие захочет человек типы данных, мы ограничем их нашими стандартными интерфейсами, которые соответсвуют своим типам данных. Так как наши интерфейсы имеют обобщеный параметр, нам так же дополнительно потребуется добавить еще один обобщающий параметр, котороый и будет передаваться в наши стандартные интерфейсы.

В конченом итоге мы получим универсальный интерфейс IGenerator, который позволяет гибко создавать генераторы, ограниченные лишь реализациями наших стандартных интерфейсов. Реализацию данного интерфейса можно увидеть на рисунке 2.1.2.7.1.

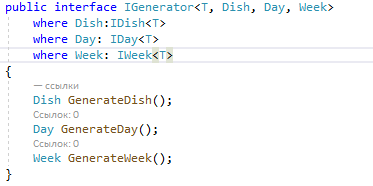


Рисунок 2.1.2.7.1 – Интерфейс генератора расписания

Так как у нас получился обобщенный интерфейс с множеством обобщённых параметров он может затруднить разработку, так как разработчик может запутаться. Создадим ему в помощь абстрактный класс DefaultAbsGenerator, в который унаследуем от наше интерфейса, и передадим туда наши стандартные интерфейсы. Его реализацию можно увидеть на рисунке 2.1.2.7.2.

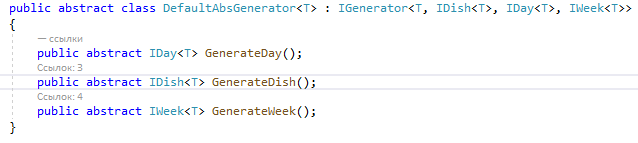


Рисунок 2.1.2.7.2 – Стандартный абстрактный класс генератора

Теперь нам лишь достаточно передать один параметр в абстрактный класс, и мы сможем переопределить все методы, что мы и сделаем. Данную реализацию назовем как DefaultGenerator.

Самым сложным в реализации будет класс конструирования блюда. Для него нам понадобится:

1. Наша база данных;
2. Правила, по которым мы будем генерировать расписание;
3. Реализации классов продуктов, блюда, недели;
4. И класс рандомайзер, для выбора продуктов блюда.

Так как у нас много полей, придется создать множество конструкторов, а также, чтобы не привязываться к определенным реализациям как типы полей укажем созданные нами стандартные интерфейсы.

2.1.2.7.1 Алгоритм составления блюда

Первым делом реализуем метод генерации блюда, так как именно на нем буду базироваться последующие алгоритмы генерации дневного и недельного рациона.

Алгоритм начнем с того, что узнаем сколько записей, содержится в наше базе данных.

Если продуктов в нашей базе данных больше 7, то берем случайное число от 5 до 7, это и будет количество продуктов, которое будет содержаться в нашем блюде. Если продуктов меньше, но больше 5, то просто берем число от 5 до количества наших записей в нашей базе данных. В остальных же случаях мы выбрасываем ошибку, и сообщаем пользователю, что данных в базе данных недостаточно.

Далее возьмем реализацию блюда, которую нам предоставил разработчик, когда создавал класс при помощи конструктора. С помощью дополнительной функции мы отчистим нужные нам поля.

Затем, мы узнали количество продуктов, возьмем это количество продуктов из нашей базы данных, конечно же случайно, но при этом будем его добавлять в наше блюдо только в том случае, если добавив его калорийность мы не будем выходить за установленные нами правила, это и будет наш использование паттерна абстрактный метод.

Суть его заключается в том, что если мы передадим другие правила или другой класс, но реализации нашего интерфейса IRules, мы все равно сможем сгенерировать расписание, и таким образом мы не зависим от конкретной реализации данного класса, а так же можем поверх уже имеющегося класса добавить паттерн построитель в виде метода SetRules и сможем генерировать расписание в зависимости от переданной реализации именно в текущем контексте программы.

Далее об алгоритме, мы считаем сумму калорий продуктов в блюде, которые мы добавили и записываем их в переменную.

У нас возникает 2 случая, если ккал меньше, чем нам нужно и когда ккал достаточно. Второй вариант заурядный, потому что в нем него не будет происходить.

Разберем первый случай.

Создадим пустой лист продуктов, для хранения продуктов, которые мы еще можем добавить. Пройдемся по всем продуктов в блюде и добавим в наш лист продуктов те, добавив которые мы не превысим калорийность нашего блюда.

Из этого у нас возникает два случая, когда подходящих продуктов не найден, и когда продукты найдены.

В первом случае мы просто берем продукт с минимальной калорийностью и добавляем его в блюдо, так как можно что бы калорийность превышалась, но нельзя что бы калорийность была меньше правил.

Во втором же случае у нас есть частный случай, когда есть только 1 продукт и общий случай, когда продуктов больше 1.

В первом частном случае мы просто берем этот продукт и добавляем его до тех пор, пока не превысим калорийность блюда.

Во втором же случае мы будем ходить по нашему списку продуктов, которые мы можем добавить, пока он не опустеет.

Создадим второй лист, в котором будем хранить продукты, которые мы можем отсеять, и сразу же проверяем есть ли в нашем наборе продуктов, продукты, которые мы можем исключить, если такие есть, то добавляем их в наш массив.

Следующим же делом, удаляем все элементы из списка продуктов, которые нам походили, и делаем проверку, если лист с продуктами стал пустым, мы выходим из цикла преждевременно с помощью команды break.

Если же продукты остались мы берем продукт с максимальной калорийностью, ищем его индекс в нашей структуре для хранения подходящих продуктов. Находим продукт с таким же именем в нашем блюде, находим его индекс в наше структуре. И добавляем его до тех пор, пока наше блюдо не станет настолько калорийным, что при еще одном добавление этого продукта, наше блюдо не станет слишком калорийным.

В конец удаляем продукт по индексу из структуры, в которой мы храним подходящие нам продукты.

Процедуру повторяем пока хранилище подходящих нам продуктов не опустеет.

2.1.2.7.2 Алгоритм составления дневного рациона

В данном методе нам надо лишь создать реализацию дня, которую нам передали, клонировать ее и отчистить поля которые нам понадобятся.

И заполнить структуру для хранения блюд, пятью уникально сгенерированными блюдами, которые соответствуют каждому приему пищи.

2.1.2.7.3 Алгоритм составления недельного рациона

В данной функции мы берем переданную нам реализацию недели, клонируем объект и отчищаем поля, которые нам понадобятся. И заполняем структуру для хранения дней, ключом, который обозначает день недели и уникально сгенерированным рационом дневного питания.

Реализовав таким образом методы мы можем сгенерировать блюдо, дневной рацион и недельный рацион, основываясь на данных которые были переданы в конструктор нашего класса.

2.2 Проектирование пользовательского интерфейса

Программу следует сделать много оконной, потому что происходит длинная инкапсуляции: неделя содержит дни, те в свою очередь блюда, а те уже продукты.

2.2.1 Графический интерфейс главного меню

Интерфейс главного меню, должен быть простой, так что будем использовать две кнопки, которые будут характерны как кнопка выхода и кнопка генерации недельного рациона. Графические вид данного окна продемонстрирован на рисунке 2.2.1.1.

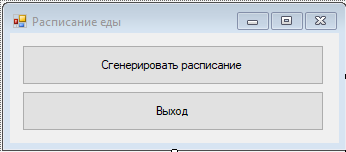


Рисунок 2.2.1.1 – Интерфейс главного меню

Нажимая на кнопку генерации расписания, будет открываться окно недельного рациона.

Копка выхода же будет закрывать окно и тем самым завершать работу приложения.

2.2.2 Графический интерфейс недели

Интерфейс недели тоже будет простой. Он будет состоять из 7 кнопок, которые характерны для каждого дня недели, копки перегенерации расписания и кнопки выхода. При появление данного окна будет создавать новый рацион и загружаться в локальную переменную, на которую уже будут ссылаться все события.

Графические вид данного окна продемонстрирован на рисунке 2.2.2.1.

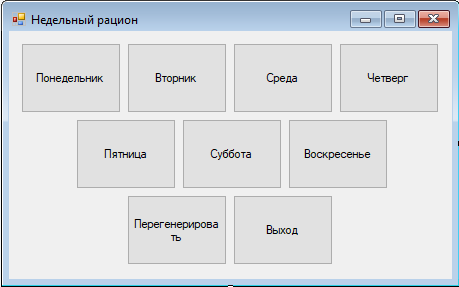


Рисунок 2.2.2.1 – Интерфейс недельного рациона

Кнопки характерные для дней недели будут отвечать за то что, если кнопка была нажата, то будут открывать окно с расписанием на конкретный день.

Кнопка для перегенерирования расписании, запускает генерацию рациона заново и добавляет новый рацион в свое хранилище.

Кнопка выхода закрывает окно.

2.2.3 Графический интерфейс рациона питания на день

Графический интерфейс будет состоять из 5 кнопок характерных для конкретного приема пищи и кнопки выхода.

Графические вид данного окна продемонстрирован на рисунке 2.2.3.1.

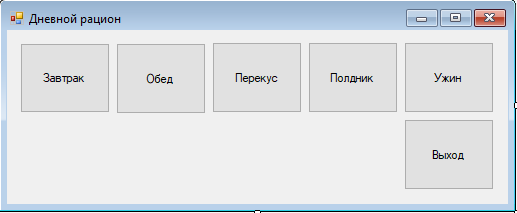


Рисунок 2.2.3.1 – Интерфейс дневного рациона

Кнопки характерные для приемов пищи отвечают, за то, что при нажатии на них будет открываться окно, в котором указано блюдо, которое нужно скушать в этот прием пищи.

Кнопка выхода закрывает окно дневного рациона.

Разработав все формы и определив события в данных графических окнах, можно интегрировать наш алгоритм в данный графический интерфейс и переходить к тестировани.

2.3 Тестирование

Тестирование выполнялось вручную, путем нажимания на все кнопки в графическом интерфейсе и сравнивая полученные значение с контрольными значениями. Такими как сумма продукты должна быть в пределах нормы, должны быть указаны все параметры продуктов и т.д.

2.4 Плюсы и минусы разработанного программного обеспечения.

Так как программа разрабатывалась на основе всех принципов построение объектно-ориентированной системы можно выделить такие плюсы как:

1. Программа очень гибкая в плане изменений;
2. Программу можно изменить лишь создав свой класс генератора и передавая в него классы, которые наследуют созданные нами интерфейсы, которые не привязаны как кой-либо конкретной реализации;
3. Все классы и интерфейсы, которые мы создали в ходе курсовой работы можно переиспользовать;
4. Существуют базовые классы, которые могу быть расширены при этом не потеряется функциональность программы;
5. Все классы находятся в разных файлах, и имеют характерные для них имена, поэтому можно легко ориентироваться в программе;
6. В ходе разработки программы, можно бы было разработать интерфейсы и отдать создание стандартных реализаций классов разным отделам, и они бы смогли их реализовать, следовательно повысилась бы скорость разработки и было бы так же просто собрать программу воедино.
7. Существует множество различных паттернов решения различных проблем, которые могу возникнуть.

Как недостатки можно выделить:

1. Необходимо понимать базовые концепции, такие как классы, наследование и динамическое связывание;
2. Многоразовое использование требует от программиста познакомиться со всеми классами, использованными в программе;
3. Проектирование классов — задача куда более сложная, чем их использование;
4. Так же реализации классов могут быть недоступны, поэтому только по интерфейсам будет не так просто понять, что делает тот или иной интерфейс в конкретных случаях;
5. Больше кода.

2.5 Руководство пользователя

Для работы требуется лишь запустить программу и по графическому интерфейсу нажимать на кнопки, реакцию на которую вы хотите получить.

Для изменения правил составления программы достаточно свою создать реализацию интерфейса правил или модернизировать существующую и передать ее в конструктор генератора расписания.

Добивать продукты в базу данных, можно указав их в файле Products.txt сохраняя структуру.

Заключение

В рамках данного курсового проект было проведение изучение принциво ООП и их применение в разработке, были выявлены плюсы и минусы разработки с применением данной парадигмы программирования.

Была разработана программа – генератор недельного рациона питания – с помощью языка программирования C#, библиотеки Windows Forms, интегрированной среды разработки Visual Studio.

Была разработан и проанализирована область знаний, к которой относится разработанное программное обеспечение.

В данной работы были разработаны классы, методы, интерфейсы, абстрактные класса, которые характерны для данной области, а также объекты, которые косвенно использовались в разработанном программном обеспечении.

В процессе разработки были досконально изучены принципы ООП и паттерны, соответствующие данной парадигме, а также использовались специальные структуры и особенности языка C#.

Результаты данной работы могут быть применены при составление недельного рациона питания в среднем на 5000 ккал, например в виде смузи. А также может служить для демонстрации принципов ООП в проектирование систем и демонстрации особенностей языка C#.

Список использованных источников

1. Архипов В. Н., Калядин В. И., Любин А. Н., Макаров А. И. Программирование на Фортране: методические указания к лабораторным работам. - М.; МГТУ «МАМИ», 2007, - 144 с
2. Калядин В. И., Макаров А. И. Основы работы на персональном компьютере: сборник лабораторных работ. - М.; МГТУ «МАМИ», 2010, - 85 с.
3. Лобанов А. С., Туманова М. Б. Решение задач на языке Visual Basic for Application: учебное пособие. - М.; МГТУ «МАМИ», 2009, - 90 с.
4. Павловская Т. A. C# Программирование на языке высокого уровня. ПИТЕР, 2007, - 432 с.

Приложение А

Листинг разработанной и представленной во второй главе проекта программы.

Листинг программы:

IDay.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using TableTime.Dish;

using TableTime.Product;

using TableTime.Rules;

namespace TableTime.Day

{

public interface IDay<T>: ICloneable

{

List<IDish<T>> Dishes { get; set; }

object Clone();

}

}

DefaultDay.cs

using System.Collections.Generic;

using TableTime.Dish;

using TableTime.Product;

namespace TableTime.Day

{

public class DefaultDay : IDay<double>

{

public List<IDish<double>> Dishes { get; set; } = new List<IDish<double>>();

private IProduct consumption;

public IProduct Сonsumption

{

get

{

return consumption;

}

set

{

consumption = value;

}

}

public DefaultDay() { }

public object Clone()

{

return this.MemberwiseClone();

}

}

}

IDish.cs

using System;

using System.Collections.Generic;

using TableTime.Product;

namespace TableTime.Dish

{

public interface IDish<T>: ICloneable

{

string Name { get; set; }

List<IProduct> Products { get; set; }

string DishName();

object Clone();

}

}

DefaultDish.cs

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using TableTime.Product;

namespace TableTime.Dish

{

public class DefaultDish : IDish<double>

{

public DefaultDish() { }

public DefaultDish(string name) { Name = name; }

public DefaultDish(params IProduct[] products)

{

if (products != null)

{

Products = products.ToList();

}

}

public DefaultDish(string name, params IProduct[] products)

{

if (products != null)

{

Products = products.ToList();

Name = name;

}

else

{

Name = name;

}

}

public List<IProduct> Products { get; set; } = new List<IProduct>();

public string Name { get; set; } = "";

public string DishName()

{

string dishName = "";

foreach(var el in Products)

{

dishName += el.Name.Substring(0, 2);

}

TextInfo myTI = CultureInfo.CurrentCulture.TextInfo;

return myTI.ToTitleCase(dishName);

}

public object Clone()

{

return this.MemberwiseClone();

}

}

}

IProduct.cs

using System;

namespace TableTime.Product

{

public interface IProduct: ICloneable

{

string Name { get; set; }

double Mass { get; set; }

double Kkal { get; set; }

double Protein { get; set; }

double Fats { get; set; }

double Carb { get; set; }

object Clone();

}

}

DefaultProduct.cs

namespace TableTime.Product

{

public class DefaultProduct : IProduct

{

public DefaultProduct()

{

Name = "";

Mass = 0.0;

Kkal = 0.0;

Protein = 0.0;

Fats = 0.0;

Carb = 0.0;

}

public DefaultProduct(string name, double mass, double kkal, double protein, double fats, double carb)

{

Name = name;

Mass = mass;

Kkal = kkal;

Protein = protein;

Fats = fats;

Carb = carb;

}

public string Name { get; set; }

public double Mass { get; set; }

public double Kkal { get; set; }

public double Protein { get; set; }

public double Fats { get; set; }

public double Carb { get; set; }

public object Clone()

{

return this.MemberwiseClone();

}

}

}

IDB.cs

namespace TableTime.DB

{

public interface IDB <T, F, O>

{

F[] Get(T key);

O Set(T key, F value);

O Update(T key, F newValue);

F[] GetAll();

}

}

AbsDb.cs

using System.Collections.Generic;

using TableTime.Product;

namespace TableTime.DB

{

public abstract class AbsDb : IDB<string, IProduct, DefaultDB>

{

protected abstract Dictionary<string, IProduct> Db { get; set; }

abstract public void FillDb(string fileName = "Products.txt");

abstract public IProduct[] Get(string key);

abstract public IProduct[] GetAll();

abstract public DefaultDB Set(string key, IProduct value);

abstract public DefaultDB Update(string key, IProduct newValue);

abstract public IProduct GetForNumber(int number);

public abstract int CountKeys();

}

}

DefaultDB.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using TableTime.Product;

namespace TableTime.DB

{

public class DefaultDB : DefaultAbsDb

{

public DefaultDB(string fileName = "Products.txt") { FillDb(fileName); }

protected override Dictionary<string, IProduct> Db { get; set; } = new Dictionary<string, IProduct>();

public override int CountKeys() => Db.Count;

public override void FillDb(string fileName)

{

using (StreamReader sr = new StreamReader(Path.GetFullPath(fileName), System.Text.Encoding.Default))

{

string line;

while ((line = sr.ReadLine()) != null)

{

line = line.Replace("\t", " ");

line = line.Replace(".", ",");

List<string> simpleLine = line.Split(' ').ToList();

IProduct product = new DefaultProduct()

{

Name = simpleLine[0],

Kkal = Convert.ToDouble(simpleLine[2]),

Protein = Convert.ToDouble(simpleLine[5]),

Fats = Convert.ToDouble(simpleLine[8]),

Carb = Convert.ToDouble(simpleLine[11]),

Mass = 100.0

};

Set(product.Name, product);

}

}

}

public override IProduct[] Get(string key)

{

if (Db.ContainsKey(key) == true)

{

return new List<IProduct>() { Db[key] }.ToArray();

}

return new List<IProduct>() { }.ToArray();

}

public override IProduct[] GetAll()

{

return Db.Values.ToArray();

}

public override IProduct GetForNumber(int number)

{

return GetAll()[number];

}

public override DefaultDB Set(string key, IProduct value)

{

Db.Add(key, value);

return this;

}

public override DefaultDB Update(string key, IProduct newValue)

{

Db[key] = newValue;

return this;

}

}

}

IRules.cs

using TableTime.Product;

namespace TableTime.Rules

{

public interface IRules

{

IProduct Сonsumption { get; set; }

}

}

DefaultRules.cs

using TableTime.Product;

namespace TableTime.Rules

{

public class DefaultRules : IRules

{

public IProduct Сonsumption { get; set; }

public DefaultRules()

{

Сonsumption = new DefaultProduct

{

Name = "Сonsumption",

Carb = 0,

Fats = 0,

Kkal = 1000,

Mass = 0,

Protein = 0

};

}

}

}

IWeek.cs

using System;

using System.Collections.Generic;

using TableTime.Day;

namespace TableTime.Week

{

public interface IWeek<T>: ICloneable

{

Dictionary<string, IDay<T>> DayOfWeek { get; set; }

object Clone();

}

}

MyDictionary.cs

using System;

using System.Collections;

using System.Collections.Generic;

namespace TableTime.Week

{

/// <summary>

/// There should be no more than 6 elements in a dict

/// </summary>

/// <typeparam name="TKey"></typeparam>

/// <typeparam name="TValue"></typeparam>

public class MyDictionary<TKey, TValue> : Dictionary<TKey, TValue>, IDictionary<TKey, TValue>

{

protected IDictionary<TKey, TValue> Dictionary;

public MyDictionary()

{

Dictionary = new Dictionary<TKey, TValue>();

}

public MyDictionary(int capacity)

{

Dictionary = new Dictionary<TKey, TValue>(capacity);

}

public MyDictionary(IEqualityComparer<TKey> comparer)

{

Dictionary = new Dictionary<TKey, TValue>(comparer);

}

public MyDictionary(int capacity, IEqualityComparer<TKey> comparer)

{

Dictionary = new Dictionary<TKey, TValue>(capacity, comparer);

}

public MyDictionary(IDictionary<TKey, TValue> dictionary)

{

Dictionary = new Dictionary<TKey, TValue>(dictionary);

}

public MyDictionary(IDictionary<TKey, TValue> dictionary, IEqualityComparer<TKey> comparer)

{

Dictionary = new Dictionary<TKey, TValue>(dictionary, comparer);

}

public virtual void Add(TKey key, TValue value)

{

if(Dictionary.Count < 8)

{

Dictionary.Add(key, value);

}

else

{

throw new OverflowException("There should be no more than 7 elements in a dict");

}

}

public virtual bool ContainsKey(TKey key)

{

return Dictionary.ContainsKey(key);

}

public virtual ICollection<TKey> Keys

{

get

{

return Dictionary.Keys;

}

}

public virtual bool Remove(TKey key)

{

return Dictionary.Remove(key);

}

public virtual bool TryGetValue(TKey key, out TValue value)

{

return Dictionary.TryGetValue(key, out value);

}

public virtual ICollection<TValue> Values

{

get

{

return Dictionary.Values;

}

}

public virtual TValue this[TKey key]

{

get

{

return Dictionary[key];

}

set

{

Dictionary[key] = value;

}

}

public virtual void Add(KeyValuePair<TKey, TValue> item)

{

Dictionary.Add(item);

}

public virtual void Clear()

{

Dictionary.Clear();

}

public virtual bool Contains(KeyValuePair<TKey, TValue> item)

{

return Dictionary.Contains(item);

}

public virtual void CopyTo(KeyValuePair<TKey, TValue>[] array, int arrayIndex)

{

Dictionary.CopyTo(array, arrayIndex);

}

public virtual int Count

{

get

{

return Dictionary.Count;

}

}

public virtual bool IsReadOnly

{

get { return Dictionary.IsReadOnly; }

}

public virtual bool Remove(KeyValuePair<TKey, TValue> item)

{

return Dictionary.Remove(item);

}

public virtual IEnumerator<KeyValuePair<TKey, TValue>> GetEnumerator()

{

return Dictionary.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

}

}

DefaultWeek.cs

using System.Collections.Generic;

using TableTime.Day;

namespace TableTime.Week

{

public class DefaultWeek : IWeek<double>

{

public DefaultWeek() { }

public DefaultWeek(Dictionary<string, IDay<double>> dayOfWeek)

{

DayOfWeek = dayOfWeek;

}

public Dictionary<string, IDay<double>> DayOfWeek { get; set; } = new MyDictionary<string, IDay<double>>();

public object Clone()

{

return this.MemberwiseClone();

}

}

}

IGenerator.cs

using TableTime.Day;

using TableTime.Dish;

using TableTime.Week;

namespace TableTime.Generator

{

public interface IGenerator<T, Dish, Day, Week>

where Dish: IDish<T>

where Day: IDay<T>

where Week: IWeek<T>

{

Dish GenerateDish();

Day GenerateDay();

Week GenerateWeek();

}

}

DefaultAbsGenerator.cs

using TableTime.Day;

using TableTime.Dish;

using TableTime.Week;

namespace TableTime.Generator

{

public abstract class DefaultAbsGenerator<T> : IGenerator<T, IDish<T>, IDay<T>, IWeek<T>>

{

public abstract IDay<T> GenerateDay();

public abstract IDish<T> GenerateDish();

public abstract IWeek<T> GenerateWeek();

}

}

DefaultGenerator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using TableTime.Day;

using TableTime.DB;

using TableTime.Dish;

using TableTime.Product;

using TableTime.Rules;

using TableTime.Week;

namespace TableTime.Generator

{

public class DefaultGenerator : DefaultAbsGenerator<double>

{

private AbsDb db = new DefaultDB();

private void SetDb(AbsDb value)

{

value.FillDb();

}

private IRules Rules { get; set; } = new DefaultRules();

private Random random { get; set; } = new Random();

private IProduct product { get; set; } = new DefaultProduct();

private IDish<double> Dish { get; set; } = new DefaultDish();

private IDay<double> Day { get; set; } = new DefaultDay();

private IWeek<double> Week { get; set; } = new DefaultWeek();

public DefaultGenerator() { }

public DefaultGenerator(AbsDb db)

{

SetDb(db);

}

public DefaultGenerator(IRules rules)

{

Rules = rules;

}

public DefaultGenerator(AbsDb db, IRules rules)

{

SetDb(db);

Rules = rules;

}

public DefaultGenerator(IProduct product)

{

this.product = product;

}

public DefaultGenerator(AbsDb db, IRules rules, Random random, IProduct product) : this(db, rules)

{

this.random = random;

this.product = product;

}

public DefaultGenerator(IRules rules, Random random) : this(rules)

{

this.random = random;

}

public DefaultGenerator(IRules rules, Random random, IProduct product) : this(rules, random)

{

this.product = product;

}

private IProduct RewriteProduct(IProduct newProduct)

{

IProduct curProduct = (IProduct)product.Clone();

curProduct.Name = newProduct.Name;

curProduct.Fats = newProduct.Fats;

curProduct.Kkal = newProduct.Kkal;

curProduct.Carb = newProduct.Carb;

curProduct.Protein = newProduct.Protein;

curProduct.Mass = newProduct.Mass;

return curProduct;

}

private IDish<double> ClearDish(IDish<double> dish)

{

var newDish = (IDish<double>) dish.Clone();

newDish.Name = "";

newDish.Products.Clear();

return newDish;

}

public override IDish<double> GenerateDish()

{

int maxValueOfProducts = db.CountKeys();

int countProducts = 0;

if (maxValueOfProducts > 7)

countProducts = random.Next(5, 7);

else if (maxValueOfProducts > 5)

countProducts = random.Next(5, maxValueOfProducts);

else

throw new Exception("Insufficient data in the database");

IDish<double> finalDish = Dish;

finalDish = ClearDish(finalDish);

for(var i = 0; i < countProducts; i++)

{

int productNumber = random.Next(0, maxValueOfProducts);

IProduct curProduct = RewriteProduct(db.GetForNumber(productNumber));

if (curProduct.Kkal + finalDish.Products.Sum( x=> x.Kkal) <= Rules.Сonsumption.Kkal)

{

finalDish.Products.Add(curProduct);

}

}

var coef = finalDish.Products.Sum( x => x.Kkal);

if (coef < Rules.Сonsumption.Kkal)

{

List<IProduct> tempList = new List<IProduct>();

foreach(var el in finalDish.Products)

{

if (coef + el.Kkal <= Rules.Сonsumption.Kkal)

{

tempList.Add(el);

}

}

if (tempList.Count == 0)

{

var minProduct = finalDish.Products.Min(x => x.Kkal);

var index = finalDish.Products.FindIndex( x => x.Kkal == minProduct);

finalDish.Products[index].Mass += 100;

finalDish.Products[index].Kkal = finalDish.Products[index].Kkal \* 2;

}

else if (tempList.Count == 1)

{

var index = finalDish.Products.FindIndex(x => x.Name == tempList[0].Name);

while (finalDish.Products.Sum(x => x.Kkal) + tempList[0].Kkal <= Rules.Сonsumption.Kkal)

{

finalDish.Products[index].Mass += 100;

finalDish.Products[index].Kkal += tempList[0].Kkal;

}

}

else

{

while (tempList.Count != 0)

{

var twoTempList = new List<string>();

foreach (var el in tempList)

{

if (finalDish.Products.Sum(x => x.Kkal) + el.Kkal >= Rules.Сonsumption.Kkal)

{

twoTempList.Add(el.Name);

}

}

foreach (var el in twoTempList)

tempList.RemoveAt(tempList.FindIndex(x => x.Name == el));

if (tempList.Count == 0)

break;

var maxProduct = tempList.Max(product => product.Kkal);

var maxKkalProductIndex = tempList.FindIndex(x => x.Kkal == maxProduct);

var index = finalDish.Products.FindIndex(x => x.Name == tempList[maxKkalProductIndex].Name);

while (finalDish.Products.Sum(x => x.Kkal) + tempList[maxKkalProductIndex].Kkal <= Rules.Сonsumption.Kkal)

{

finalDish.Products[index].Mass += 100;

finalDish.Products[index].Kkal += tempList[maxKkalProductIndex].Kkal;

//finalDish.Products[index].Kkal = finalDish.Products[index].Kkal \* (finalDish.Products[index].Mass / 100);

}

tempList.RemoveAt(maxKkalProductIndex);

}

}

}

return finalDish;

}

private IDay<double> ClearDay(IDay<double> day)

{

var tempDay = (IDay<double>)day.Clone();

tempDay.Dishes.Clear();

return tempDay;

}

public override IDay<double> GenerateDay()

{

IDay<double> tempDay = new DefaultDay();

tempDay = ClearDay(tempDay);

for (var i = 0; i < 5; i ++)

{

var dish = GenerateDish();

tempDay.Dishes.Add(dish);

tempDay.Dishes[i].Name = dish.DishName();

}

return tempDay;

}

public override IWeek<double> GenerateWeek()

{

var tempWeek = (IWeek<double>) Week.Clone();

tempWeek.DayOfWeek.Clear();

List<string> nameDays = new List<string>() { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };

foreach(var el in nameDays)

{

tempWeek.DayOfWeek.Add(el, GenerateDay());

}

return tempWeek;

}

}

}

Program.cs

using System;

using System.Windows.Forms;

namespace TableTime

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

Form1.cs

using System;

using System.Windows.Forms;

using TableTime.Generator;

using TableTime.Week;

namespace TableTime

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

DefaultAbsGenerator<double> generator = new DefaultGenerator();

IWeek<double> week = generator.GenerateWeek();

Form weekForm = new WeekForm(week);

weekForm.Show();

}

private void button3\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

DishForm.cs

using System;

using System.Windows.Forms;

using TableTime.Dish;

namespace TableTime

{

public partial class DishForm : Form

{

private string Format = "Продукт: {0} Ккал {1}, Белки {2}, Жиры {3}, Углеводы {4}, Масса {5}" + Environment.NewLine;

public IDish<double> Dish { get; set; }

public DishForm()

{

Dish = new DefaultDish();

InitializeComponent();

}

public DishForm(IDish<double> dish)

{

Dish = dish;

InitializeComponent();

showDish();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

private void showDish()

{

textBox1.Text = "";

textBox1.Text += string.Format("Блюдо: {0}" + Environment.NewLine, Dish.Name);

foreach(var el in Dish.Products)

{

textBox1.Text += string.Format(Format, el.Name, el.Kkal, el.Protein, el.Fats, el.Carb, el.Mass);

}

}

}

}

DayForm.cs

using System;

using System.Windows.Forms;

using TableTime.Day;

namespace TableTime

{

public partial class DayForm : Form

{

public IDay<double> Day { get; set; }

public DayForm()

{

Day = new DefaultDay();

InitializeComponent();

}

public DayForm(IDay<double> day)

{

Day = day;

InitializeComponent();

}

private void ShowDish(int index)

{

Form dishForm = new DishForm(Day.Dishes[index]);

dishForm.Show();

}

private void button1\_Click(object sender, EventArgs e)

{

ShowDish(0);

}

private void button5\_Click(object sender, EventArgs e)

{

ShowDish(4);

}

private void button6\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button2\_Click(object sender, EventArgs e)

{

ShowDish(1);

}

private void button3\_Click(object sender, EventArgs e)

{

ShowDish(2);

}

private void button4\_Click(object sender, EventArgs e)

{

ShowDish(3);

}

}

}

WeekForm.cs

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using TableTime.Generator;

using TableTime.Week;

namespace TableTime

{

public partial class WeekForm : Form

{

public IWeek<double> Week { get; set; }

private List<string> NameDays { get; set; } = new List<string>() { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };

public WeekForm()

{

Week = new DefaultWeek();

InitializeComponent();

}

public WeekForm(IWeek<double> week)

{

Week = week;

InitializeComponent();

}

private void ShowDay(int index)

{

Form dayForm = new DayForm(Week.DayOfWeek[NameDays[index]]);

dayForm.Show();

}

private void button1\_Click(object sender, EventArgs e)

{

ShowDay(0);

}

private void button9\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button8\_Click(object sender, EventArgs e)

{

var generator = new DefaultGenerator();

Week = generator.GenerateWeek();

}

private void button2\_Click(object sender, EventArgs e)

{

ShowDay(1);

}

private void button3\_Click(object sender, EventArgs e)

{

ShowDay(2);

}

private void button4\_Click(object sender, EventArgs e)

{

ShowDay(3);

}

private void button5\_Click(object sender, EventArgs e)

{

ShowDay(4);

}

private void button6\_Click(object sender, EventArgs e)

{

ShowDay(5);

}

private void button7\_Click(object sender, EventArgs e)

{

ShowDay(6);

}

}

}