Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Поволжский государственный университет телекоммуникаций и информатики»

Кафедра информатики и вычислительной техники

Задание к экзамену по дисциплине

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

|  |  |
| --- | --- |
| Составил: | Е. А. Часов |
| Ревизия | 2019/1.3.2 |
| Язык программирования | Java, JDK 1.8+ |
| Поток | 2РПИС, 2ПО |
| Предыдущие редакции | 2018/1.2.2  2018/1.1.2  2017/0.2 |

**Самара**

**2019г.**

**Содержание**

[Примечания и рекомендации 3](#_Toc536190885)

[1 Входные данные 17](#_Toc536190886)

[2 Задания на выполнение 20](#_Toc536190887)

[2.1 Импорт 20](#_Toc536190888)

[2.2 Методы поиска 20](#_Toc536190889)

[2.3 Методы экспорта 21](#_Toc536190890)

[2.4 Способы представления 21](#_Toc536190891)

[2.5 Поиск родственников 21](#_Toc536190892)

[2 Структура проекта и требования к решению 22](#_Toc536190893)

Примечания и рекомендации

В заданиях звездочкой (\*) помечены пункты, которые не являются обязательными, но их реализация будет явным преимуществом.

Общие рекомендации по Code-style:

Избегайте объявления объектов в циклах или объявления для хранения промежуточных значений без их последующего удаления.

Обработку исключений лучше делать содержательной, как минимум требуется запись в лог или вывод ошибки на консоль.

Всегда оценивайте, насколько код является оптимальным с точки зрения количества операций (асимптотической сложности), потребления памяти и использованию ресурсов системы (например, работа с жестким диском).

Создание методов

Метод – отдельная функция или процедура, выполняющая в идеальном случае только одну задачу. Самая важная цель создания методов – «Снижение сложности программы». Создайте метод для сокрытия информации, чтобы о ней можно было не думать. Конечно при написании метода думать о ней придется, но после этого можно забыть о деталях и использовать метод, не парясь о его внутренней работе. Остальные цели – минимизация кода, облегчение сопровождения программы и снижение числа ошибок – также хороши, но без абстрагирующей силы методов сложные программы было бы невозможно охватить умом.

Цели создания методов

1 Формирование понятной абстракции

Выделение фрагмента кода в удачно названный метод – один из лучших способов документирования его цели. То есть, вместо того, чтобы при чтении программы разбираться в строках кода, вы, прочитав имя вызванного метода, сразу поймете, что происходит. То есть удачные методы облегчают чтение и понимание кода, а также снижают его сложность.

2 Предотвращение дублирование кода

Несомненно, самая популярная причина создания метода. Включение похожего кода в два и более методов указывает на ошибку декомпозиции. Уберите повторяющийся фрагмент из методов, поместите его общую версию в базовый класс и создайте специализированные методы в подклассах. В другом случае вы можете выделить общий код в отдельный метод и вызвать его первоначальных методов. В результате программа станет компактней, изменять ее станет проще, так как в случае чего вам нужно будет изменить только один метод. Код станет надежнее, потому что для его проверки нужно будет проанализировать только один фрагмент. Изменения будут реже приводить к ошибкам, поскольку вы не сможете по невнимательности внести в идентичные фрагменты программы чуть различающиеся изменения.

3 Сокрытие очередности действий

Если в программе необходимо чтобы метод В вызывался после метода А, скройте эту деталь реализации – поместите вызовы в метод С и вызывайте его.

4 Сокрытие операций над указателями

Операции над указателями не отличаются удобочитаемостью и легкостью понимания и часто являются источниками ошибок. Изолировав такие операции в методах, вы сможете сосредоточиться на их сути, а не на механизме манипуляций над указателями. Кроме того, выполнение операций над указателями в одном месте облегчает проверку правильности кода. Если же вы найдете более эффективный тип данных, чем указатели, изменения затронут лишь несколько методов.

5 Улучшение портируемости

Использование методов изолирует непортируемый код, явно определяя фрагменты, которые придется изменить при портировании приложения. В число непортируемых аспектов входят нестандартные возможности языка, зависимости от оборудования и операционной системы и т.д.

6 Упрощение сложный булевых проверок (глубокая вложенность условных операторов)

Понимание сложных булевых проверок редко требуется для понимания пути выполнения программы. Поместив такую проверку в метод, вы сможете упростить код, потому что: а) детали проверки будут скрыты и б) описательное имя метода позволит лучше охарактеризовать суть проверки. Создание отдельного метода для проверки подчеркивает ее значимость. Это, в свою очередь, мотивирует программистов сделать детали проверки внутри метода более удобочитаемыми. В результате и основной путь выполнения кода и сама проверка становятся более понятными.

7 Упрощение глубокой вложенности циклов

По аналогии со сложными условиями, если в программе встречаются вложенные циклы – разумнее вынести их в отдельный метод, а описательное имя метода позволит не заморачиваясь на повторяющихся вычислениях понять их суть. Тем самым упрощается понимание кода и уменьшается его сложность.

8 Сокрытие глобальных данных

Всем известно, что дорога в ад программирования вымощена глобальными переменными. Поэтому не стоит давать доступ к данным, используемым разными классами и модулями напрямую. Потому что в случае изменения формата хранения данных или их типа, придется менять все компоненты, использующие эту глобальную переменную. Разумнее использовать 2 открытых метода начинающиеся с имен get (для получения значения) и set (для установки значения), с помощью которых обеспечивается доступ к закрытой глобальной переменной. И в случае чего вам нужно будет изменить только эти два метода (на крайний случай добавить перегрузку в эти методы).

Перегрузка методов – механизм, позволяющий методу с одним и тем же именем иметь разные реализации, в зависимости от числа и типов входных аргументов.

По возможности старайтесь избегать глобальных данных.

**Правила именования методов**

Имя метода должно ясно описывать все, что он делает. При выборе имени можно воспользоваться следующими рекомендациями:

1 Описывайте все, что метод выполняет

Имя метода должно содержать все выходные данные и все побочные эффекты. Например, если метод генерирует отчет о событии и открывает для записи выходной log-файл, то имя generateEventReport() не является адекватным. generateEventReportAndOpenLogFile() – адекватное имя, но слишком длинное и несуразное. А причина этому – побочный эффект. Выход из положения – создавать ясные методы без побочных эффектов, преследующие одну цель. Логичнее было бы создать 2 метода:

generateEventReport

writeLogFile()

2 Избегайте невыразительных и неоднозначных глаголов

Некоторые глаголы могут описывать практически любое действие. Имена методов вроде

performServices()

outputUser()

processInput()

doSomethingWithInput()

dealWithOutput()

ничего не говорят о работе методов. В лучшем случае можно догадаться что они имеют какое-то отношение к сервисам, пользователям, вводу и выводу соответственно.

Метод сам по себе может быть спроектирован очень хорошо, но абсолютная невыразительность его имени заставит копаться в методе. С другой сторон, несуразное имя может быть попыткой описания несуразного метода. В этом случае необходимо пересмотреть метод и все родственные методы.

3 Не используйте для дифференциации имен методов номера

Номера в методах как правило не говорят о различиях предоставляемых методами абстракций, поэтому их не следует применять. Например,

method1()

method2()…

4 Длина имени

Поскольку главная задача имени метода – это как можно более ясное и понятное описание сути метода, то имя метода может иметь любую длину, удовлетворяющую этой цели. Но не забывайте, если имя получается слишком длинное, скорее всего следует пересмотреть метод и разделить его на несколько (что может затронуть и родственные методы).

5 Для именования функции лучше использовать описание возвращаемого значения

Поскольку функция возвращает значение, то это следует должным образом отразить в ее имени. Так, имена

cos()

version()

errorStatus()

isEmpty()

ясно указывают что возвращают функции.

6 Для именования процедуры используйте выразительный глагол, дополняя его объектом

В случае объектно-ориентированных языков имя объекта в имя процедуры включать не нужно, потому что объекты и так входят в состав вызовов:

printDocument(), writeLog()

Document.Print(), Log.write().

Document.PrintDocument() – избыточное имя.

7 Используйте антонимы

Применение антонимов поддерживает согласованность имен, что облегчает чтение кода. Популярные антонимы:

add/remove increment/decrement open/close get/set

begin/end insert/delete show/hide old/new

create/destroy lock/unlock source/target push/pull

first/last min/max start/stop

get/put next/previous up/down

8 Определяйте конвенции именования часто используемых операций

Сохраняйте однообразность именования во всем проекте, чтобы не возникало путаницы в понимании операций. Существуют разные виды операций, которые могут описываться схожими или одинаковыми именами – их также следует разделять, используя конвенции именования.

9 Объем метода

Следует относиться с осторожностью к методам, объемом более 200 строк. При определении длины метода следует ориентироваться на сложность – глубина вложенности управляющих конструкций, число переменных, число точек принятия решений, число комментариев, необходимых для объяснения метода.

Параметры метода

1 Передавайте параметры в порядке «входные значение – изменяемые значения – выходные значения»

Или, по крайней мере, придерживайтесь единого порядка во всех методах разрабатываемой программы. Некоторые языки на синтаксическом уровне определяют такой порядок входных параметров, другие – иной, а есть языки не контролирующие порядок в принципе.

2 Если несколько методов используют похожие параметры, передавайте их в согласованном порядке.

Вместо

Copy(sourceFile, destinationFile)

move(destinationFile, sourceFile)

логично использовать

copy(sourceFile, destinationFile)

move(sourceFile, destinationFile)

3 Используйте все параметры

Если вы передаете параметр в метод, используйте его, в противном случае – удалите параметр из интерфейса метода. Как показывают исследования, наличие неиспользуемого параметра соответствует более высокому уровню ошибок. Кроме того, не используемые параметры снижают эффективность программы (хоть и не намного). Исключение – в процессе отладки и тестирования вы можете передавать неиспользуемые параметры, но на этапе реализации – исключите их.

4 Передавайте переменные статуса и кода ошибки последними

Многие методы могут вносить изменения в переменные статуса и кода ошибки, которые используются в отладке, тестировании и сопровождении программ. Они второстепенны по отношению к главной цели метода и являются исключительно выходными параметрами, поэтому логично располагать их в конце.

5 Не используйте параметры метода в качестве рабочих переменных

Использовать передаваемые в метод параметры как рабочие переменные – опасно. Потому что в процессе работы метода вы измените исходное значение этого параметра, а оно может понадобиться в другой части программы. Лучше создать локальную переменную, и присвоить ей значение входного параметра. Такая операция подчеркнет тот факт, что значение поступает в метод из вне. Входные параметры лучше передавать по значению (большинство языков понимают это объявление входного параметра).

6 Добавляйте предположения о параметрах в комментариях к методу и в местах его вызовов

Если предполагается, что передаваемые в метод данные должны иметь определенные характеристики – сразу же документируйте это. Пишите комментарии не дожидаясь завершения работы над методом – к тому времени как правила много забывается =)). Примеры предположений о параметрах:

— вид параметров (являются ли они исключительно входными, изменяемыми или исключительно выходными)

— единицы измерения (м, см, дБ, Гц и т.д)

— смысл кодов статуса и ошибок, если для их представления не используется перечисление.

— диапазоны допустимых значений

— специфические значения, которые никогда не должны передаваться методу

7 Ограничиваете число параметров примерно 7

7 – магическое число. Психологические исследования показали, что люди, как правило, не могут следить более чем за 7 элементами информации одновременно. В высокоуровневых языках вы можете передать составной тип данных, состоящий из 15 полей, но рассматривать их всех как один параметр. В низкоуровневых, возможно придется передать все 15 полей как отдельные параметры. Передача большого числа параметров – увеличивает сопряжение. Если вы передаете одни и те же параметры в разные методы, логично будет сгруппировать эти методы и данные в класс.

8 Использование конвенции именования формальных параметров методов

Если нужно провести различие между входными, изменяемыми и выходными параметрами, сформулируйте соответствующие конвенции именования. Например, добавить соответствующие префиксы к именам.

9 Используйте явно именованные параметры

Многие языки программирования позволяют явно сопоставить формальные параметры с фактическими. То есть имена фактических и формальных параметров будут одинаковы. Это делает применение параметров более ясным и помогает избежать ошибок сопоставления параметров.

Создание классов позволяет снизить сложность за счет:

— инкапсуляции (сокрытия) деталей реализации. То есть доступ к вспомогательным методам и атрибутам, необходимым для реализации основных (открытых) методов не предоставляется остальным классам. Например, когда мы говорим о двери, вы предоставляете информацию о цвете, размерах двери, о дверной ручке, а не о волокнах дерева, из которого изготовлена дверь и шурупах, которыми прикручена ручка к двери.

— ограничения влияния изменений. Разделение классов, реализующих бизнес-правила, ввод-вывод, работу с оборудованием и ОС.

— сокрытие глобальных данных за интерфейсом класса. Например, для доступа к данным использовать методы доступа, а сами данные оставить закрытыми для доступа.

2. Глобальные переменные — скрыть, а для доступа к ним использовать методы.

3. Для класса следует определять 5-9 методов и 5-9 членов данных.

4. Избегайте создания божественных классов – которые все могут и умеют.

5. Минимизируйте сотрудничество класса с другими классами – вызовы методов объектов других классов и обращение с данными других классов. Правило Деметры – А может вызвать любые свои методы, если он создает В, он может использовать его методы, ему не следует вызывать методы объектов, возвращаемых объектом В.

Управление сложностью

В настоящее время (да в принципе уже лет 30-40) ПО настолько сложное, что охватить его целиком программисту не получится. Над крупными проектами трудятся команды программистов, при этом каждая команда отвечает только за одну часть проекта, даже не подозревая о том, как работают другие части разрабатываемого ПО. Поэтому в современных проектах стараются достичь слабого сопряжения компонентов ПО. Сопряжение характеризует силу связи класса или метода с другими классами или методами.

Иными словами, ПО проектируют и реализуют таким образом, чтобы компоненты были как можно более не зависимыми, чтобы разработчик мог вносить изменения с как можно меньшим числом последствий (то есть чтоб изменение одного компонента не повлекли за собой изменение других компонентов). Как правило разделают следующие подсистемы:

— подсистема бизнес правил. Бизнес-правила – это законы, директивы, политики и процедуры, реализуемые в программной системе. Например, в случае создания системы расчета заработной платы бизнес-правилами могли бы стать директивы налоговой службы, определяющей разные виды налогов, нормативные документы организации, регулирующие размер заработной платы и премий и т.д.

— подсистема пользовательского интерфейса. Изоляция компонентов пользовательского интерфейса в отдельной подсистеме позволяет изменять его, не влияя на остальную программу. Как правило эта система включает несколько подсистем, отвечающих за интерфейс командной строки, графический интерфейс GUI, работа с меню, управление окнами, справочная система и т.д.

— подсистема доступа к данным. Она нужна чтобы скрыть детали реализации доступа к данным, чтобы большая часть программы не нуждалась в знании «грязных» подробностей операций на над низкоуровневыми структурами и могла работать с данными в терминах бизнес-проблемы. Эти подсистемы концентрируют операции на данными в одном месте и снижают вероятность ошибок при работе с данными, а также позволяют легко изменять структуру данных без изменения большей части программы.

Переменные

Рассмотрим только именование, ибо остальные аспекты связаны с синтаксисом языков.

1. Самый важный принцип именования переменных – имя должно полно и точно описывать сущность, предоставляемую переменной

Одним из эффективных способов выбора хорошего имени является формулирование сути переменной в словах. Оптимальным именем переменной часто оказывается само это высказывание. Так как оно является полным описанием сущности, его нельзя спутать с чем-либо другим. Например, переменную, представляющую число студентов в группе ПО-11 можно было бы назвать numberOfStudentsOnThePO111Group, переменную, представляющую число сидячих мест для инвалидов в автобусе numberOfSeatsForInvalidsInTheBus. Для хранения числа студентов, делающий домашнюю работу по Программированию можно было бы ввести переменную numberOfStudentDoingHomework, и ее значение было бы 0=)). Переменную хранящую размер n-го измерения массива можно было бы назвать lengthNDimension, a не N или LND.

Особенность предложенных имен состоит в том, что, во-первых, их легко расшифровать (фактически, расшифровывать и не нужно – достаточно их прочитать), во-вторых имена слишком велики, чтобы быть практичными. Имена переменных должны быть максимально конкретными. Например, что хранит в себе переменная

current – текущее что? дату, давление, сахар в крови, номер пары?

date – дату, но какую именно? Текущую дату, день независимости или день рождения песика Томаса?

cd – слишком короткое и не описательное. Если аббревиатура, то какая – cool down или current date? А может это обозначение химического элемента Кадмий?

currentDate – четко и понятно, что в ней хранится текущая дата. А в чем смысл текущей даты ясно из самой задачи – то есть из контекста ее применения.

2. При подборе имени лучше ориентироваться на проблему

Хорошее имя чаще всего описывает проблему, а не ее решение. То есть оно в большей степени выражает что а не как. Если же имя описывает некоторый аспект вычислений, а не проблемы, имеет место обратное. Предпочитайте имена, характеризующую саму проблему. Запись данных о сотруднике, можно было бы назвать inputRec или employeeData. inputRec – компьютерный термин, выражающий идеи ввода данных и записи. employeeData относится к проблемной области, а не к миру компьютеров. В случае битового поля, определяющего статус принтера, имя bitFlag более компьютеризированно, чем printerReady.

3. Длина имени переменной

Понятно, что слишком короткие имена – не информативны, а слишком длинные – не практичны (усложняют визуальное восприятие и в некоторых случаешь, если среда разработки не поддерживает intely sence – долго печатать + ошибки в именах при печати). В соответствии с различными исследованиями, рекомендуется длина переменных 9-16 символов. Одним из критериев выбора длины имени также может служить «значимость» переменной. Короткое имя переменной говорит о том, что она является второстепенной и имеет ограниченную область действия (например, переменная i – используется в нескольких строках кода в качестве индекса массива или счетчика цикла и не играет никакой роли вне этих строк). Можно придерживаться такого подхода – редко используемым или глобальным переменным присваивать длинные имена, а локальным — короткие. Однако, для повышения надежности ПО, лучше избегать короткие имена. Истина лежит по середине – старайтесь подбирать имена, ясно и полно описывающие суть переменной. Получилось сравнительно короткое имя – хорошо, длинное – не беда.

4. Добавляйте спецификаторы вычисляемых значений

Многие программы включают переменные, содержащие вычисляемые значения суммы, средние величины, максимумы и т.п. Дополняя такое имя спецификатором вроде Total, Sum, Average, Max, Min, Default, Record, String или Pointer, укажите его в конце. Во-первых, самая значимая часть имени останется в начале и будет читаться первой. Во-вторых, избежите путаницы с наличием в программе двух переменных, вроде totalEmployeeSalary и employeeSalaryTotal. В-третьих, используя эту конвенцию во всей программе, избежите путаницы, а согласованность имен облегчит чтение и сопровождение программы.

5. Антонимы

Часто используемые антонимы в именах переменных

Begin/end first/last locked/unlocked min/max old/new

opened/closed up/down visible/invisible source/target source/destination

6. Именование индексов циклов

Не присваивайте имена i, j, k ничему, кроме простых циклов (1-2 действия за каждый проход). В остальных случаях используйте описательные имена индексов.

7. Имена временных переменных

Часто для хранения промежуточных результатов вычислений используются временные переменные, с именами вроде temp, buf. Лучше использовать вместо них описательные имена, поскольку это упрощает чтение кода, и уменьшат затраты времени на расширение фрагмента кода в случае необходимости.

8. Имена булевых переменных

Присваивайте булевым переменным имена, подразумевающие значение true или false. Для этого можно использовать следующие имена:

done – признак завершения цикла или другой операции. Значение False – до начала операции, True – по завершении операции.

error – признак возникновения ошибки. False – если все в порядке, True – в противном случае.

found – признак обнаружение чего-либо (определенного значения, файла, объекта). False — значение не обнаружено, True – как только значение найдено.

success, ok – Признак успешного завершения операции. False – операция завершилась неудачей, True – операция выполнена успешно.

Наиболее популярный вариант – начинать имена булевых переменных (равно как и методов, возвращающих логическое значение) с префикса is:

isFound, isDone.

Используйте утвердительные имена булевых переменных – такой подход не вызовет замешательства. Например, какая строчка будет верна:

if ( notFound )

или

if (not notFound)

9 Именование констант

Имя константы должно характеризовать абстрактную сущность, предоставляемую константой, а не конкретное значение.

Плохие имена констант FIVE, CURRENT\_DATE.

Хорошие имена констант DAYS\_IN\_APRIL\_TOTAL, CARS\_COUNT\_MAX

10 Практические рекомендации

Для разных языков – будут свои нюансы, перечислим те рекомендации, которые применимы к большинству языков программирования.

10.1 Форматируйте имена так, чтобы их легко было читать. Выберите один из 2-х способов – либо пишите все строчными, а первые буквы слов – заглавными (стиль camalCase), или все строчными, а между словами подчеркивание.

hages\_count\_total

pagesCountTotal

Относительно первой буквы – используйте стиль того языка программирования, на котором реализуется проект. Но как правило имена переменных, атрибутов класса и методов начинаются со строчной буквы, а имена, классов и интерфейсов– с заглавной.

10.2 Имена типов (не относится к классам)

1 в имени типа используйте префикс T\_ или t\_.

2 имя типа должно быть более общим, а имя переменной – более конкретное. Например, тип t\_emploeeBase, а переменная generalMotorsEmployeeBase.

10.3 Именование констант

Либо используйте префикс c\_, либо пишите имя только заглавными буквами, разделяя слова символами подчеркивания.

10.5 Имена перечислений

Имена самих перечислений, можно начинать с префикса e\_, E\_ или использовать только заглавные буквы. А для именования каждого члена (значения) можно использовать префикс – имя перечисления (множества). Перечисление:

enum e\_apples {

apples\_golden,

apples\_pinkLady,

apples\_spartak,

apples\_whiteJuice

}

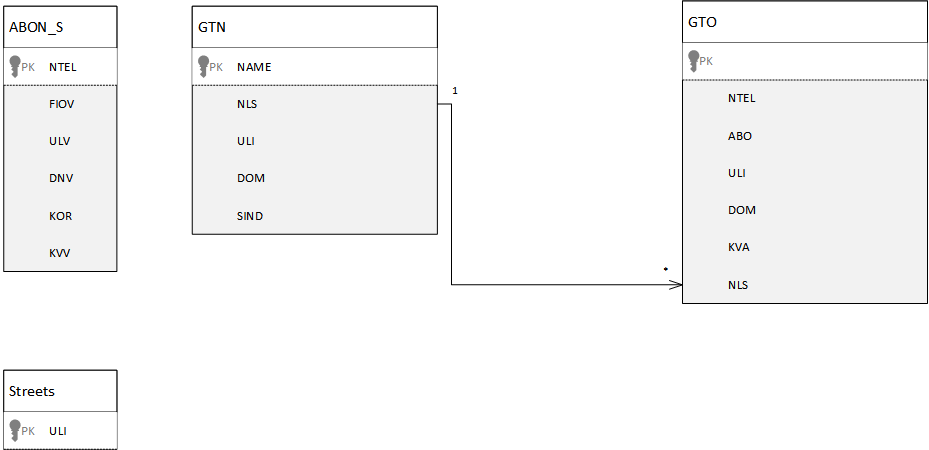
1 Входные данные

Есть проект, разработка которого велась десятилетие назад. Возникла задача портировать его на новые технологии с сохранением максимального количества данных и связей между ними. Проект представляет собой приложение-клиент для базы данных жителей города. БД проектировалась и хранилась в разных форматах: разные форматы Excel, Acces и FoxPro. В результате экспорта из проприетарных форматов данные оказались в текстовом виде.

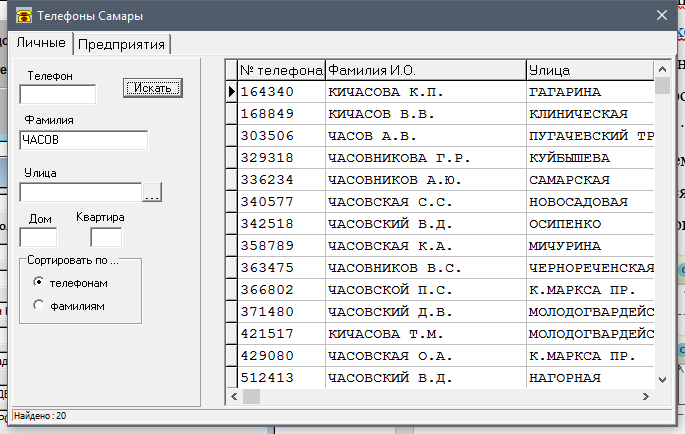
Содержимое папки city – база данных жителей города. Записей в базе больше 10^6 элементов (город-миллионник). Так как база данных была экспортирована в текстовый формат из разных файлов отличающегося формата, то содержащаяся в них информация может иметь разное отображение. Информация разбита на несколько файлов:

* ins.txt, ins1.txt – информация непосредственно о жителях
* pred.txt, pred1.txt – информация о предприятиях, связанных с жителями
* standart.txt, standart1.txt – информация о регионах
* streets.txt, streets1.txt –информация об улицах населенных пунктов

Поскольку информация была экспортирована из разных частей разных БД, то информация может дублироваться. К тому же экспортом данных и форком на новые технологии начали заниматься люди, которые не являются изначальными разработчиками приложения. Вследствие этого требуется провести частичный реверс-инжиниринг БД: установить значение полей в таблицах и связи между ними.

В результате анализа было выяснено следующее. Часть базы, хранящая информацию о телефонных номерах, содержит следующие таблицы и связи: 

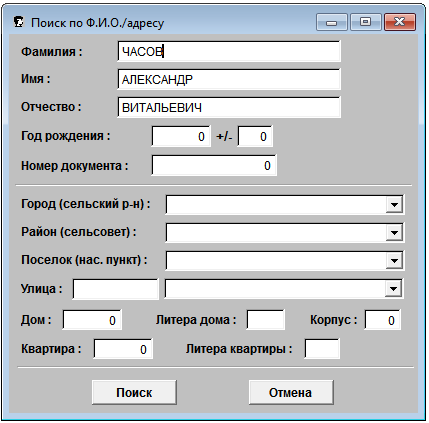
Приложение для этой БД выглядело следующим образом:



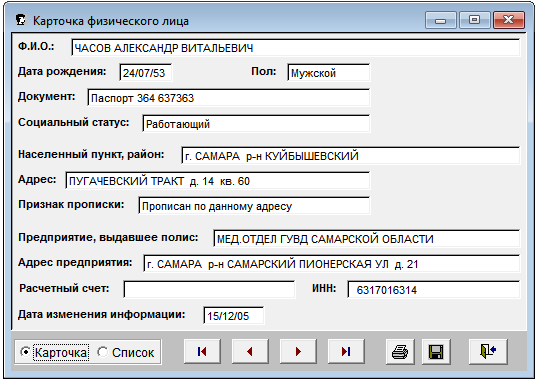
Существовала также версия в виде БД Acces, формат данных тот же.

Другая часть БД не содержит телефонных номеров и приложение для нее выглядело так:

Форма запроса по различным параметрам:



Выбранная запись из списка результата запроса:



Таблицы и **выявленные** связи для этой БД отображены в файле Fiz\_DB.xlsx

2 Задания на выполнение

2.1 Импорт

1) Реализовать импорт данных из файлов в систему, удалив дублирующие записи из набора. Структуры данных выбирать целесообразно и самостоятельно.

1.1) Если имеются необходимые компетенции, разрешается реализовать импорт данных с разработкой схемы БД под управлением SQLite.

2) По возможности в результате импорта в структуре, хранящей БД, не должно быть ссылок на «внешние ключи» - вместо них должна находиться информация, которая соответствует этому ключу; либо, если информация импортирована раздельно – обеспечить «бесшовное» получение информации о жителе города.

3) Для тех жителей или юридических лиц, у которых в базе данных есть телефонный номер, добавить его к информации о жителе.

4) При импорте информации об улицах населенных пунктов добавить к названию и номеру улицы случайное значение в диапазоне от 50 до 20 000.

5) Для числовых данных использовать числовой формат, для даты – формат даты (Date, LocalDate, Calendar, LocalDateTime(since Java 8)) и пр.

2.2 Методы поиска

Реализовать методы запроса информации о человеке по различным критериям:

* ФИО
* Дата рождения
* Адрес
* Телефон
* Номер документа (любого)
* И т.д.

В случае совпадения параметров запроса у нескольких человек отображать список объектов.

Учесть, что текстовый поиск (например, по ФИО или адресу) может быть частичным, т.е. нужно искать вхождение в подстроке.

Также реализовать фильтрацию по одному из критериев.

2.3 Методы экспорта

Реализовать методы экспорта и импорта объектов с помощью сериализации. Способ сериализации (в бинарный объект, XML/JSON или текстовый файл) предоставляется выбрать самостоятельно.

Реализовать методы экспорта объектов, полученных в результате запроса по выбранным параметрам.

2.4 Способы представления

Реализовать эффективные алгоритмы сортировки БД по полям.

2.5 Поиск родственников

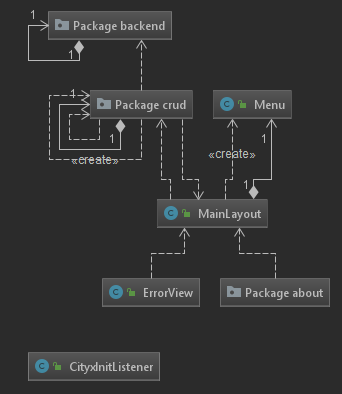
Реализовать метод поиска возможных родственных связей между записями в БД. Очевидная подсказка: можно проверить соответствие отчества имени отца, совпадение фамилии, проверить корректность дат рождения (родители не могут быть моложе детей), возможно совпадает адрес прописки. Метод должен однозначно указывать предполагаемую родственную связь (А отец Б, С мать Б, В брат Б, Б и В сын/дочь А и Б и т.д.)

3 Структура проекта и требования к решению

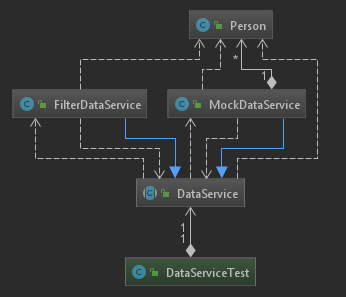
Код решения должен быть расположен в соответствующем пакете шаблона проекта и собран в соответствии с его конфигурацией. Шаблон проекта находится в папке cityx. Проект выполнен в виде двух модулей – backend и ui.

Для облегчения реализации отображения данных и работы с ними в проекте находится прототип приложения с классами. На данный момент структура классов и модулей следующая:

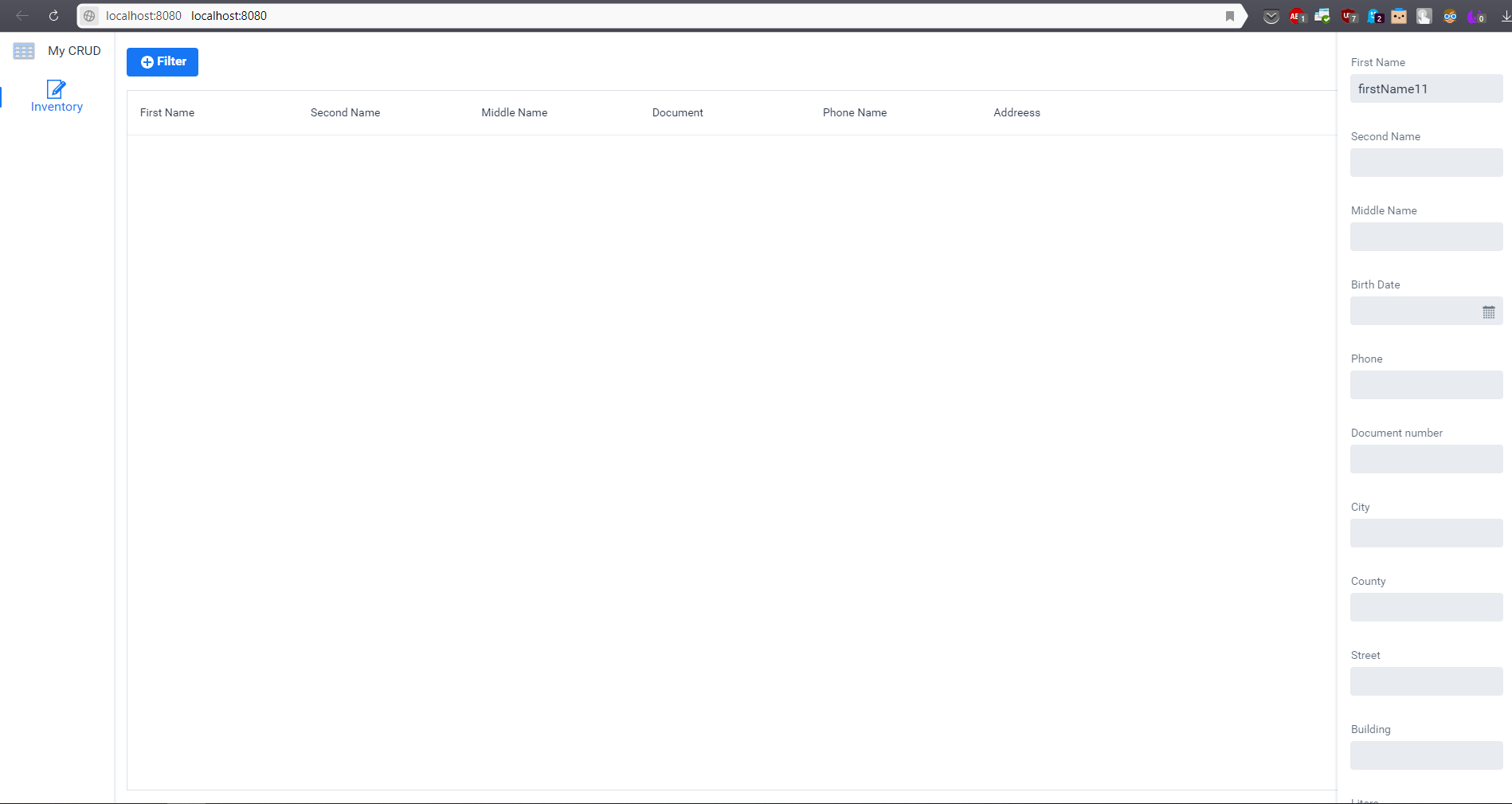
UI:



Backend:



Код в проекте не является всецело правильным и единственно верным и может редактироваться на усмотрение студента с учетом целесообразности. Ответственность за редактирование кода в модуле ui возлагается целиком на студента. Интерфейс выполнен в виде веб-приложения на Vaadin и выглядит следующим образом:



Код на момент скачивания будет содержать строки, генерирующие тестовый рандомный набор значений для тестирования графического интерфейса, методы решений заданий не реализованы.

Обработку текстовых файлов, запросы к БД и прочие вещи, связанные с обработкой данных предлагается реализовать в модуле backend. Для его использования нужно реализовать решение в соответствии с вызываемым API в проекте.

Оценивается полнота реализации решения, корректность и стиль кода, правильность реализации алгоритмов, подход к решению, архитектура итогового проекта, способность обосновать принятые решения и объяснить работу разработанных классов и методов.

ОС и ресурсы компьютеры на рабочих местах могут не выдерживать нагрузки и работы с количеством записей в 1,5\*10^6 элементов. В качестве решения предлагаются следующие варианты:

* Увеличение потребляемой памяти JVM при помощи ключей
* «Динамическая» сериализация с последующей динамической подгрузкой данных из сериализованных объектов. В этом случае при импорте информации из текстовых файлов следует выгружать объекты на диск через каждые 400-500 тысяч записей. В методах обращения к данным учесть это и реализовать перебор и загрузку всех сериализованных объектов для обеспечения поиска информации по всем импортированным записям.

Документ не является исчерпывающим и в случае возникновения вопросов можно обратиться к преподавателю, Церковь, военкомат, администрацию госдепа или завязать с фантазиями о карьере разработчика.