Содержание:

Введение

Данная курсовая работа посвящена предмету «объектно-ориентированное программирование» и теме «анализ структуры бензозаправки и построение диаграммы классов».

Целью этой работы является закрепление навыков полученых при изучении дисциплины «Объектно-ориентированное программирование» путём анализа сущности и построения её диаграммы классов.

В данной работе объектом исследования является бензозаправка.

Нам предстоит проанализировать бензозаправку и выделить наиболее значимые её составляющие. Также дать описания классам находящимся в ней и их связям и на основе полученных результатов построить полную диаграмму классов, наглядно показывающей основные компоненты и их взаимодействие между собой в данной структуре.

В ходе работы мы будем полагаться на знания полученные при изучении «Объектно ориентированного программирования».

Глава 1. Классы в ООП

Одним из наиболее важных понятий ООП является класс. Класс представляет собой дальнейшее развитие концепции типа и объединяет в себе задание не только структуры и размера переменных, но и выполняемых над ними операций. Объекты в программе всегда являются экземплярами того или иного класса (аналогично переменным определенного типа).

Объект является представителем или экземпляром класса. Во время выполнения программы объекты взаимодействуют друг с другом, вызывая методы характерные для определенного класса. Класс определяет интерфейс с окружающим миром, посредством которого можно взаимодействовать с отдельными объектами. Все представители данного класса аналогичны друг другу. Каждый конкретный класс

имеет свои особенности поведения и характеристик, определяющих этот класс.

Каждый класс должен иметь имя; если имя слишком длинно, его можно сократить или увеличить сам значок на диаграмме. Имя каждого класса должно быть уникально в содержащей его категории. Для некоторых языков, в особенности - для C++ и Smalltalk, мы должны требовать, чтобы каждый класс имел имя, уникальное в системе. [1]

Класс - необходимое, но недостаточное средство декомпозиции. Когда система разрастается до дюжины классов, можно заметить группы классов, связанные внутри, и слабо зацепляющиеся с другими. Мы называем такие группы категориями классов. [2]Многие объектно-ориентированные языки не поддерживают это понятие. Следовательно, выделение обозначений для категорий классов позволяет выразить важные архитектурные элементы, которые не могли быть непосредственно записаны на языке реализации Среда программирования Smalltalk поддерживает концепцию категорий классов. Однако, в Smalltalk категории классов не имеют семантического содержания: они существуют только для более удобной организации библиотеки классов. В C++ категории классов связаны с концепцией компонент (Страуструп), они еще не являются чертой языка, хотя включение в него семантики пространства имен рассматривается.

Классы и категории классов могут сосуществовать на одной диаграмме. Верхние уровни логической архитектуры больших систем обычно описываются несколькими диаграммами, содержащими только категории классов. Категории классов. Категории классов служат для разбиения логической модели системы. Категория классов - это агрегат, состоящий из классов и других категорий классов, в том же смысле, в котором класс - агрегат, состоящий из операций и других классов. Каждый класс системы должен "жить" в единственной категории или находиться на самом верхнем уровне системы.

В отличие от класса, категория классов не имеет операций или состояний в явном виде, они содержатся в ней неявно в описаниях агрегированных классов.[3]

Иногда полезно на значке категории перечислить некоторые из содержащихся в ней классов. "Некоторые", потому, что зачастую категории содержат довольно много классов, и перечислять их все было бы хлопотно, да это и не нужно.

Так же, как список атрибутов и операций на значке класса, список классов в значке категории представляет сокращенный вид ее спецификации. Если мы хотим видеть на значке категории больше классов, мы можем его увеличить. Можно удалить

разделяющую черту и оставить в значке только имя категории. Категория классов представляет собой инкапсулированное пространство имен. По аналогии с квалификацией имен в C++, имя категории можно использовать для однозначной квалификации имен содержащихся в ней классов и категорий. Например, если дан класс A из категории B, то его полным именем будет A::B. Таким образом, как будет обсуждаться далее, для вложенных категорий квалификация имен простирается на произвольную глубину. Некоторые классы в категории могут быть открытыми, то есть экспортироваться для использования за пределы категории. Остальные классы могут быть частью реализации, то есть не использоваться никакими классами, внешними к этой категории.

Для анализа и проектирования архитектуры это различие очень важно, так как позволяет разделить обязанности между экспортируемыми классами, которые берут на себя общение с клиентами, и внутренними классами в категории, которые, собственно, выполняют работу. На самом деле, во время анализа закрытые аспекты категории классов можно опустить. По умолчанию все классы в категории определяются как открытые, если явно не указано противное.

Ограничение доступа будет обсуждаться ниже. Категория может использовать невложенные категории и классы. С другой стороны, и классы могут использовать категории. Для единообразия мы обозначаем эти экспортно-импортные отношения так же, как отношение использования между классами (см. рис. 5-4). Например, если категория A использует категорию B, это означает, что классы из A могут быть наследниками, или содержать экземпляры, использовать или быть еще как-то ассоциированы с классами из B. [4]

[5]Классы могут быть физически вложены в другие классы, а категории классов - в другие категории и т.д. Обычно это нужно для того, чтобы ограничить видимость имен. Вложение соответствует объявлению вложенной сущности в окружающем ее контексте. Обычно вложение классов является тактическим решением проектировщика, а вложение категорий классов - типично стратегическое архитектурное решение. В обоих случаях необходимость в использовании вложения на глубину более одного-двух уровней встречается крайне редко.

1.1 Диаграмма классов в ООП

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их кооперaций,

атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Целью создания диаграммы классов является графическое представление статической структуры декларативных элементов системы (классов, типов и т. п.) Она содержит в себе также некоторые элементы поведения (например — операции), однако их динамика должна быть отражена на диаграммах других видов (диаграммах коммуникации, диаграммах состояний). Для удобства восприятия диаграмму классов можно также дополнить представлением пакетов, включая вложенные.

При представлении сущностей реального мира разработчику требуется отразить их текущее состояние, их поведение и их взаимные отношения. На каждом этапе осуществляется абстрагирование от маловажных деталей и концепций, которые не относятся к реальности (производительность, инкапсуляция, видимость и т. п.) Классы можно рассматривать с позиции различных уровней. Как правило, их выделяют три основных: аналитический уровень, уровень проектирования и уровень реализации

- на уровне анализа класс содержит в себе только набросок общих контуров системы и работает как логическая концепция предметной области или программного продукта.
- на уровне проектирования класс отражает основные проектные решения касательно распределения информации и планируемой функциональности, объединяя в себе сведения о состоянии и операциях.
- на уровне реализации класс дорабатывается до такого вида, в каком он максимально удобен для воплощения в выбранной среде разработки; при этом не воспрещается опустить в нём те общие свойства, которые не применяются на выбранном языке программирования.

Диаграмма классов является ключевым элементом в объектно-ориентированном моделировании. На диаграмме классы представлены в рамках, содержащих три компонента.

В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом. Имена классов начинаются с заглавной буквы. Если класс абстрактный — то его имя пишется полужирным курсивом.

Посередине располагаются поля (атрибуты) класса. Они выровнены по левому краю и начинаются с маленькой буквы.

Нижняя часть содержит методы класса. Они также выровнены по левому краю и пишутся с маленькой буквы.

Диаграмма классов показывает классы и их отношения, тем самым представляя логический аспект проекта. Отдельная диаграмма классов представляет определенный ракурс структуры классов. На стадии анализа мы используем диаграммы классов, чтобы выделить общие роли и обязанности сущностей, обеспечивающих требуемое поведение системы. На стадии проектирования мы пользуемся диаграммой классов, чтобы передать структуру классов, формирующих архитектуру системы.

Диаграммы классов верхнего уровня, содержащие только категории классов, представляют архитектуру системы в самом общем виде. Такие диаграммы чрезвычайно полезны для визуализации слоев и разделов системы. Слой обозначает набор категорий классов одного уровня абстракции. Таким образом, слои представляют набор категорий классов, так же как категории классов - это кластеры классов. Слои обычно нужны, чтобы изолировать верхние уровни абстракции от нижних. Разделы обозначают связанные (каким-либо образом) категории классов на разных уровнях абстракции. В этом смысле слои представляют собой горизонтальные срезы системы, а разделы - вертикальные.[6]

1.2 Отношения между классами

Классы редко бывают изолированы; напротив, как объяснялось в главе 3, они вступают в отношения друг с другом. Виды отношений показаны на рис. 5-4: ассоциация, наследование, агрегация (has) и использование.

При изображении конкретной связи ей можно сопоставить текстовую пометку, документирующую имя этой связи или подсказывающую ее роль. Имя связи не обязано быть глобальным, но должно быть уникально в своем контексте.[7] Значок ассоциации соединяет два класса и означает наличие семантической связи между ними. Ассоциации часто отмечаются существительными, например Employment (место работы), описывающими природу связи. Класс может иметь ассоциацию с самим собой (так называемая рефлексивная ассоциация). Одна пара классов может иметь более одной ассоциативной связи.

Возле значка ассоциации вы можете указать ее мощность, используя синтаксис следующих примеров:

1 - В точности одна связь

N - Неограниченное число (0 или больше)

0..N - Ноль или больше

1..N - Одна или больше

0..1 - Ноль или одна

3..7 - Указанный интервал

1..3, 7 - Указанный интервал или точное число

Обозначение мощности пишется у конца линии ассоциации и означает число связей между каждым экземпляром класса в начале линии с экземплярами класса в ее конце. Если мощность явно не указана, то подразумевается, что она не определена. Обозначения оставшихся трех типов связи уточняют рисунок ассоциации дополнительными пометками. [8]

Это удобно, так как в процессе разработки проекта связи имеют тенденцию уточняться. Сначала мы заявляем о семантической связи между двумя классами, а потом, после принятия тактических решений об истинных их отношениях, уточняем эту связь как наследование, агрегацию или использование. Значок наследования, представляющего отношение "общее/частное", выглядит как значок ассоциации со стрелкой, которая указывает от подкласса к суперклассу.

В соответствии с правилами выбранного языка реализации, подкласс наследует структуру и поведение своего суперкласса. Класс может иметь один (одиночное наследование), или несколько (множественное наследование) суперклассов. Конфликты имен между суперклассами разрешаются в соответствии с правилами выбранного языка. Как правило, циклы в наследовании запрещаются. К наследованию значок мощности не приписывается. [9]

Значок агрегации обозначает отношение "целое/часть" (связь "has") и получается из значка ассоциации добавлением закрашенного кружка на конце, обозначающем агрегат. Экземпляры класса на другом конце стрелки будут в каком-то смысле частями экземпляров класса-агрегата. Разрешается рефлексивная и циклическая

агрегация. Агрегация не требует обязательного физического включения части в целое. Знак использования обозначает отношение "клиент/сервер" и изображается как ассоциация с пустым кружком на конце, соответствующем клиенту.

Эта связь означает, что клиент нуждается в услугах сервера, то есть операции класса-клиента вызывают операции класса-сервера или имеют сигнатуру, в которой возвращаемое значение или аргументы принадлежат классу сервера.

Глава 2. Анализ и описание структуры сущности

Целью данной работы была выбрана Бензозаправка.

Для начала произведём анализ структуры данного объекта и выделим наиболее значимые его части.

Бензозаправку можно поделить на 2 обобщенных части, это саму «заправку» и «магазин». [10]

Поскольку мы не собираемся слишком сильно дробить объекты входящие в нее будем описывать лишь обощенные объекты.

На «заправке» находятся колонки с помощью которых производится заправка, работник осуществляющий её. Также к «заправке» можно отнести хранилище горючего из которого топливо поступает в бензоколонку. Поэтому мы разделим «заправку» на 3 объекта: бензоколонка, заправщик и бензохранилище.

«Магазин» также является совокупностью нескольких объектов. В «магазине» можно выделить «кассу» она выполняет функцию триггера в оказании услуг, работника который выполняет различные обязанности внутри помещения, торговое помещение область отведённая под магазин и склад от наличия товаров в котором зависит работа торгового помещения.

Также можно выделить клиента в качестве отдельного объекта хоть он и не является частью объекта поскольку он является ключевой фигурой в работе бензозаправке. Клиент взаимодействует с «заправкой» и «магазином».

Попробуем представить её с помощью объектно-ориентированного программирования(ООП).

В ООП главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Поэтому представим заправку как совокупность некоторых классов связанных между собой.[11]

На заправке можно выделить такие основные классы как:

- Класс « Заправка» в нем содержится несколько вложенных классов:
    - Класс «Заправщик» человек производящий работу на заправке.
    - Класс «Бензоколонка» аппарат с помощью которого класс «Заправщик» заправляет автомобиль по сигналу от класса «Касса».
    - Класс «Бензохранилище» цисцерна в которой хранится бензин и из которой он поступает в бензоколонку.
- Класс «Магазин» в нем содержатся несколько вложенных классов таких как :
    - Класс «Касса» через него проходят все денежные операции касающиеся предоставляемых услуг.
    - Класс «Работник» это человек работающий в магазине.
    - Класс «Торговое помещение» небольшое помещение продающее небольшой ассортимент товаров.
    - Класс «Склад» помещение использующееся для хранения товаров и предметов.
- Класс «Клиент» это человек который пришел на бензозаправку для приобретения определённых услуг

Глава 3. Построение диаграммы классов

После определения классов составим из них диаграмму классов.

Нам нужно рассмотреть каждый класс по отдельности и выяснить его возможные связи с другими классами.

Но для начала дадим определение различным видам связей классов.

Зависимость

Зависимость обозначает такое отношение между классами, что изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.

Ассоциация

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому. Является общим случаем композиции и агрегации.[12]

Например, класс Человек и класс Школа имеют ассоциацию, так как человек может учиться в школе. Ассоциации можно присвоить имя «учится в».

Двойные ассоциации представляются линией без стрелочек на концах, соединяющей два классовых блока. Ассоциации более высокой степени имеют более двух концов и представляются линиями, один конец которых идёт к классовому блоку, а другой к общему ромбику. В представлении однонаправленной ассоциации добавляется стрелка, указывающая на направление ассоциации.

Ассоциация может быть именованной, и тогда на концах представляющей её линии будут подписаны роли, принадлежности, индикаторы, мультипликаторы, видимости или другие свойства.

Агрегация

Диаграмма классов, показывающая Агрегацию между двумя классами

Агрегация — это разновидность ассоциации при отношении между целым и его частями. Как тип ассоциации агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).[13]

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют агрегацию по ссылке, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет.

Графически агрегация представляется пустым ромбом на блоке класса, и линией, идущей от этого ромба к содержащемуся классу.

Композиция

Композиция — более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Графически представляется как и агрегация, но с закрашенным ромбиком.

Различия между композицией и агрегацией

Приведём наглядный пример. Комната является частью квартиры, следовательно здесь подходит композиция, потому что комната без квартиры существовать не может. А, например, мебель не является неотъемлемой частью квартиры, но в то же время, квартира содержит мебель, поэтому следует использовать агрегацию. [14]

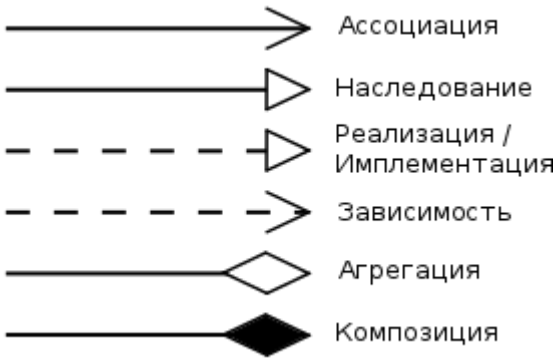

Рис 1. Обозначение связей на диаграмме классов.[15]

Приступим к описанию возможных связей классов друг с другом.

Класс «Магазин» имеет несколько классов являющихся его частью это: «Касса», «Склад», «Работник», «Торговое помещение». Соответственно все эти классы имеют связи Композиция по отношению к классу «Магазин».

Тот же тип связи имеют и классы «Заправщик», «Бензоколнка», «Бензохранилище» к классу «Заправка» поскольку они также являются частью класса «Заправка».

Класс «Клиент» имеет связь Ассоциация с классами «Магазин» и «Заправка» поскольку он будет передвигаться и взаимодействовать с различными классами входящими в них.

Класс «Работник» имеет связь Ассоциация с классами «Касса», «Склад», «Торговое помещение», поскольку работник будет перемещаться и взаимодействовать с любым из них в процессе работы. Например принять оплату за услугу или учет

товара на складе и т.д.

Класс «Торговое помещение» имеет связь Зависимость с классом «Склад» поскольку если товары на складе закончатся, то класс не сможет функционировать, поскольку торговать будет нечем.

Класс «Заправщик» имеет связь Зависимость с классом «Касса» поскольку только после оплаты денег на кассе заправщик приступит к работе с классом «Бензоколонка». Также у него имеется связь Ассоциация с классами «Бензоколонка» и «Бензохранилище» поскольку он будет передвигаться и работать с этими двумя классами. Например заправка автомобиля или закачка бензина в хранилище.

Класс «Бензоколонка» имеет связь Зависимость с классом «Бензохранилище» поскольку если в хранилище закончится горючее то он не сможет функционировать.

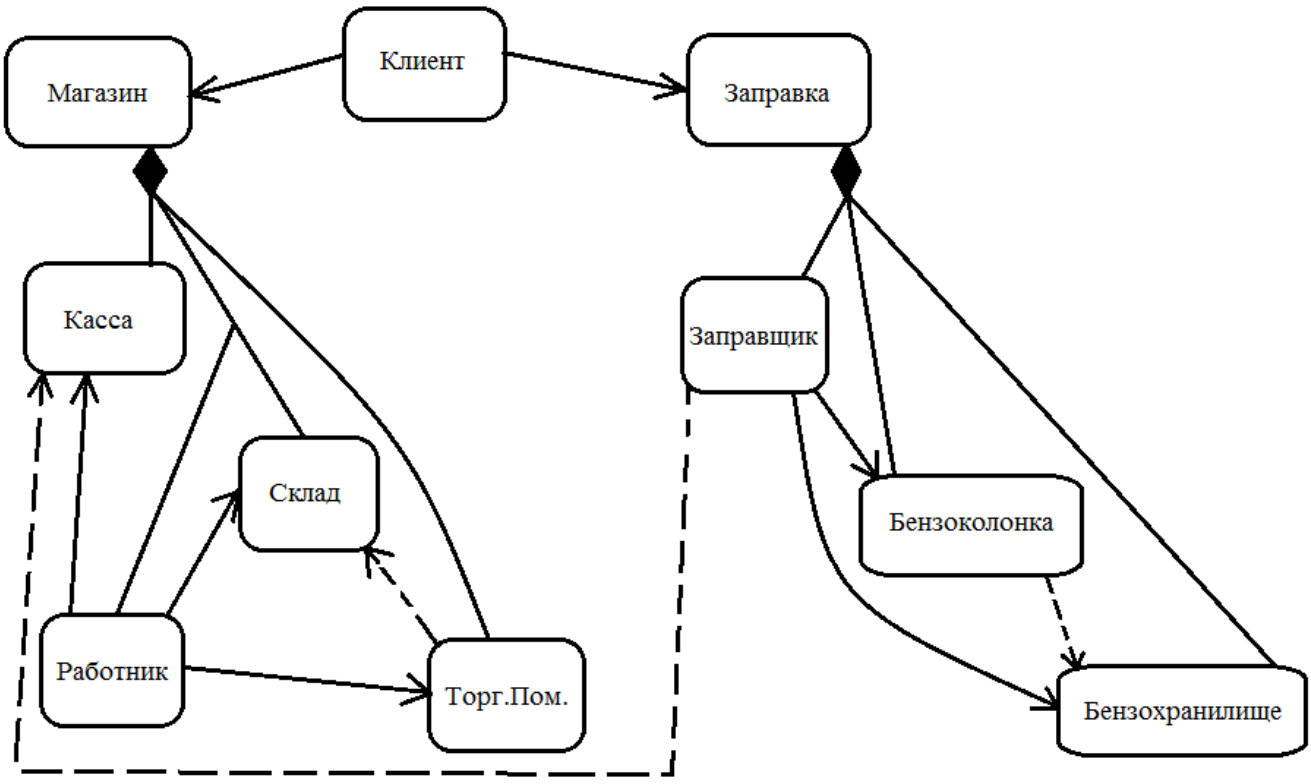

Рис 2. Диаграмма классов Бензозаправки

Заключение

В ходе работы мы использовали полученные навыки и знания в сфере объектно-ориентированного программирования. Выделили в бензозаправке совокупности объектов и представили их в виде классов. Далее установили связи между ними и исходя из этого составили диаграмму классов.

Построение диаграммы классов позволяет наглядно изобразить программу, организацию или процесс в виде схемы классов и их взаимосвязей. Благодаря этому можно быстро разобраться в алгоритме, функциях и назначении тех или иных объектов сущности, что способствует экономии времени и ресурсов. Этот способ широко используется в программировании, бизнесе и организации процессов.

Список литературы

1. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд. - СПб.: Питер, 2011.

2. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010

3. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч; пер. с англ. И. Романовский, Ф. Андреев. - М.: Бином, 2012.

4. Себеста, Р.У. Основные концепции объектно-ориентированного программирования / Р.У. Себеста. - 5-е изд. - М.: Издательский дом «Вильямс», 2011.

5. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — (Серия «Библиотека программиста»).

6. Почувствуй класс / Мейер Б.; пер. с англ. под ред. В.А. Биллига.—М.: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011.

7. Дж. Кьоу Объектно-ориентированное программирование / Дж. Кьоу, М. Джеанини. - М.: Питер, 2015.

1. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд. - СПб.: Питер, 2011. ↑

2. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

3. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

4. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

5. Почувствуй класс / Мейер Б.; пер. с англ. под ред. В.А. Биллига.—М.: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011. ↑

6. Дж. Кьоу Объектно-ориентированное программирование / Дж. Кьоу, М. Джеанини. - М.: Питер, 2015. ↑

7. Себеста, Р.У. Основные концепции объектно-ориентированного программирования / Р.У. Себеста. - 5-е изд. - М.: Издательский дом «Вильямс», 2011. ↑

8. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — (Серия «Библиотека программиста»). ↑

9. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч; пер. с англ. И. Романовский, Ф. Андреев. - М.: Бином, 2012. ↑

10. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — (Серия «Библиотека программиста»). ↑

11. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч; пер. с англ. И. Романовский, Ф. Андреев. - М.: Бином, 2012. ↑

12. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

13. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

14. Фридман, А.Л. Основы объектно-ориентированной разработки программных систем / А.Л. Фридман. - М.: Финансы и статистика, 2010 ↑

15. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч; пер. с англ. И. Романовский, Ф. Андреев. - М.: Бином, 2012. ↑