# Объектно-ориентированное проектирование

Для удобства всех программистов можно условно разделить на создателей классов (тех, кто определяет новые типы данных) и прикладных программистов («потребителей», использующих типы данных в своих приложениях).   
*Прикладные программисты* собирают библиотеки классов и используют их для ускорения разработки приложения.   
*Создатель класса* стремится построить класс, который предоставляет прикладному программисту доступ лишь к тому, что необходимо для его работы, и скрывает все остальное. Почему?   
Скрытые аспекты класса не могут использоваться прикладными программистами, поэтому создатель класса может свободно изменять их, не беспокоясь о том, как эти изменения отразятся на других. Скрытая часть класса обычно представляет чувствительные «внутренности» объекта, которые могут быть легко испорчены прикладным программистом по неосторожности или от недостатка информации, поэтому скрытие реализации уменьшает количество ошибок в программе.

Прежде чем начать программирование, необходимо выполнить проектирование программы, то есть дать ответы всего на три вопроса.

• Какие функции должна выполнять система?   
• Какие объекты будут использованы при реализации? (Как разбивать проект на составляющие)   
• Какими интерфейсами они будут обладать? (Какие сообщения должны отправляться каждому объекту).

Даже если у вас не будет ничего, кроме представлений об объектах и их интерфейсах, можно переходить к написанию программы. По различным причинам вам могут потребоваться другие описания и документы, но меньшим обойтись невозможно.

Процесс разработки объектно-ориентированных программ делится на пять фаз с дополнительной нулевой фазой, на которой просто принимается решение об использовании той или иной структуры процесса.

Фаза 0 — составление плана

Сначала следует решить, из каких этапов будет состоять процесс. На первый взгляд звучит просто (собственно, все формулировки такого рода будут звучать просто), но нередко подобные решения принимаются уже после начала программирования.  
 Если вы руководствуетесь планом «Поскорее взяться за программирование, а дальше ориентироваться по обстановке» — что ж, замечательно (при решении очевидных проблем иногда достаточно и этого). По крайней мере, осознайте, что ваш план звучит именно так.   
В результате планирования проект делится на части, с которыми удобнее работать и которые выглядят не столь устрашающе. Наличие промежуточных контрольных точек помогает сосредоточиться и направляет усилия программиста к этим контрольным точкам, не позволяя ему зациклиться на одной цели — «завершении проекта» (в нашем случае - получение хорошей оценки на экзамене).   
Следовательно, даже если ваш план сводится к тому, чтобы поскорее взяться за программирование, все равно стоит пройти следующие фазы, задать и ответить на некоторые вопросы.

*Фаза 1* - *формулировка задачи (что делать).*

Функциональные требования определяют ключевые особенности системы и дают представление о некоторых основных классах, которые будут использоваться в вашей программе. В сущности, набор функциональных требований состоит из содержательных ответов на следующие вопросы:   
• Кто будет использовать эту систему?   
• Какие операции смогут выполнять операторы?   
• Как оператор будет выполнять с помощью этих операций ту или иную функцию?   
• Какие еще возможны варианты, если та же операция будет выполняться кем-то другим или у того же оператора появится другая цель (анализ вариантов)?   
• Какие проблемы с системой могут возникнуть при выполнении операции (анализ исключений)?

Например, если вы проектируете ИС «Библиотека», то функциональные требования будут описывать действия ИС в любой возможной ситуации. Каждая из таких ситуаций называется сценарием. Сценарий можно рассматривать как вопрос, начинающийся со слов: «Как поступит система, если...?»   
Например:   
- Как поступит система, если студент заказывает книгу, а в библиотеке нет свободных экземпляров этой книги?  
- Как поступит система, если студент заказывает книгу, а у него на руках есть книга, срок сдачи которой истек?

Диаграммы функциональных требований должны быть как можно проще, чтобы вы раньше времени не увязли в подробностях реализации.   
Можно ограничиться списком с несложными *UML*-диаграммами прецедентов, одна из которых приведена на рисунке 3.1.   
Фигурки на этом рисунке изображают «операторов» — обычно это люди или какие-либо независимые агенты (например, другая компьютерная система). Прямоугольник изображает границы системы, а овалы — описания отдельных функций, выполняемых системой. Линии, соединяющие операторов с операциями, изображают взаимодействия. В данном случае важна не конкретная реализация системы, а ее структура с точки зрения пользователя, поэтому даже для сложной системы функциональные требования не обязаны быть излишне сложными.

Эта фаза завершается разработкой документа «Техническое задание», содержание отдельных пунктов которого может быть в дальнейшем скорректировано по взаимному согласию заказчика и разработчика.

*Фаза 2 — как делать*

В этой фазе создается архитектура, описывающая классы и их взаимодействия.  
Признаком завершения фазы 2 является законченное описание объектов и их интерфейсов... или по крайней мере их большей части — некоторые из них остаются неучтенными и выявляются только во время фазы 3. Это вполне нормальное явление. Важно лишь, чтобы все объекты были рано или поздно выявлены. Хорошо, когда они обнаруживаются на ранней стадии процесса, но ООП обладает достаточной структурной устойчивостью, так что более позднее обнаружение не принесет особого вреда.   
Результат представляется в виде *UML*-диаграммы взаимодействия классов.

Возможно, вам также потребуется описать структуры данных в тех системах или подсистемах, в которых доминирующая роль отводится данным (например, при работе с базами данных).

*Фаза 3 — построение ядра (программирование системы классов)*

В этой фазе первоначальный набросок архитектуры преобразуется в компилируемый исполняемый код, который можно протестировать. Но самое главное, что прототип должен либо подтвердить, либо опровергнуть исходную архитектуру.

Фактически вы строите каркас, который будет наращиваться при последующих итерациях. Также при этом выполняется первая попытка системной интеграции и первое тестирование, а заказчик получает представление о том, как будет выглядеть его система и как идет работа. Одной из составляющих разработки системы является сравнение промежуточного результата с анализом требований и системными спецификациями (в той форме, в которой они существуют). Убедитесь в том, что результаты тестирования соответствуют всем требованиям и сценариям. Когда ядро системы заработает стабильно, можно двигаться дальше и добавлять новые возможности.

*Шаг 4 — итеративный перебор сценариев*

Получив работоспособное ядро, вы начинаете добавлять к нему новые возможности.   
Каждое функциональное требование представляет собой совокупность связанных функциональных возможностей, которые встраиваются в систему как единое целое.  
Итеративность позволяет обнаружить и решить критически важные проблемы на более ранней стадии: у заказчика появляется возможность откорректировать начальную постановку задачи, работа программиста становится более творческой, а сам проект лучше управляется.



Рисунок 3.1. UML-диаграмма прецедента «Выдача книги студенту»

**Проработка предметной области задания**

В ЛВП2 студенты в составе бригады выполняют индивидуальное задание , которая кратко формулируется так: “Разработать систему классов для …”. Могут быть даны незначительные пояснения и уточнения к теме, хотя студент должен выполнить проработку (фактически проектирование), в основном, самостоятельно. Для этого нужно представить задачу или задачи, для которых система класса может быть использована. Это позволит сформулировать требования к классам: их свойства (характеристики) и поведение (методы класса). Далее выполняется словесное описание этих требований и программное описание классов по правилам языка программирования.

В каждом задании разрабатывается минимум один контейнерный класс и один элементный класс (см. ниже).

Первым шагом в выполнении задания ЛВП2 должно быть **осмысление** поставленной задачи, и, в частности, выделение новых типов объектов, для которых нужно разработать обобщенные описания объектов в виде классов на языке С++. Нужно определить свойства объектов каждого класса и его поведение. Другими словами, необходимо первоначально выделить основные данные для объектов класса и перечень методов/функций, которые эти данные изменяют. На первом этапе это делается на абстрактном уровне.

Для начала проектирования является важным определение понятия **предметной области**. Под *предметной областью* понимается совокупность понятий, объектов реального мира, их свойств, а также что с этими объектами можно делать в программе и делается в жизни. Для одной и той же системы классов может быть выделено много различных предметных областей, которые, в конечном счете, зависят от решаемой (поставленной) задачи автоматизации. Поэтому первым шагом нужно задать одну или несколько предметных областей, для которых будет разрабатываться система классов.

Одновременно с разработкой классов должна быть разработана проектная документация в виде: диаграмм классов тестового примера, уже на этом этапе нужно думать о сдаче проекта, поэтому продумывается и программа методика испытаний системы классов на соответствие требованиям ТЗ. Выделяется и разрабатывается также документация для эксплуатации программного продукта (Руководство пользователя).

Диаграмма классов для разрабатываемого в курсе ПКШ макета учебной СУБД имеет следующий вид:



Если в задании на ЛВП2 требуются действия, которые неудобно делать с помощью интерфейса класса DBTable из библиотеки dbmsLib, то для выполнения этих действий в проекте нужно разработать свою систему классов, а dbmsLib использовать для хранения результатов обработки в БД.