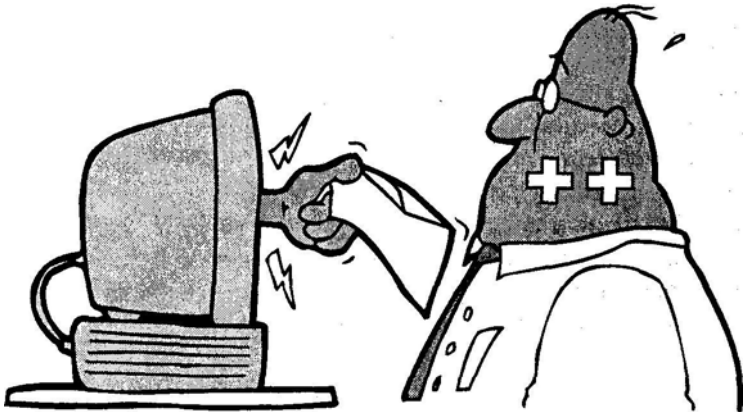

0 ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

В настоящее время объектно-ориентированное программирование (ООП) является доминирующим стилем при создании больших программ. Рассмотрим основные этапы эволюции структурного подхода в программировании, которые помогают лучше понять взаимосвязь структурного подхода, модульного программирования и ООП.

0.1 Структурный подход в программировании

Первые программы для цифровых вычислительных машин редко превышали объем 1 кбайт. С той поры существенно изменилась архитектура и технические характеристики программируемых вычислительных средств (ВС), чрезвычайно снизилась стоимость хранения, пересылки и обработки 1 байта информации. Объемы используемых программ и программных систем измеряются не только десятками килобайтов, но и сотнями мегабайтов. Вместе с тем удельная стоимость создания программ (нормированная объемом программы) до последнего времени менялась мало. Более того, с ростом объема программы удельная стоимость ее создания могла нелинейно возрастать. Было обнаружено, что время создания сложных программ пропорционально квадрату или даже кубу объема программ. Поэтому не удивительно, что одним из основных факторов, определяющих развитие технологии программирования, является снижение стоимости проектирования и создания программных продуктов (ПП), или борьба со сложностью программирования.

Другими важнейшими факторами, влияющими на эволюцию методов проектирования и создания ПП, являются:

- изменение архитектур вычислительных средств (ВС) в интересах повышения производительности, надежности и коммуникативности;
- упрощение взаимодействия пользователей с ВС и интеллектуализация ВС.

Действие двух последних факторов, как правило, сопряжено с ростом сложности программного обеспечения ВС. *Сложность* представляет неотъемлемое свойство программирования и программ, которое *проявляется* во времени и стоимости создания программ, в объеме или длине текста программы, характеристиках ее логической структуры, задаваемой операторами передачи управления (ветвления, циклы, вызовы подпрограмм и др.).

Можно выделить 5 следующих источников сложности программирования:

1) решаемая задача;
2) язык программирования;
3) среда выполнения программы;
4) технологический процесс коллективной разработки и создания ПП;
5) стремление к универсальности и эффективности алгоритмов и типов данных.

От свойства сложности нельзя избавиться, но можно изменять характеристики его проявления путем управления или организации.

В программировании широко используется фундаментальный принцип управления сложными

системами, который известен человеку с глубокой древности — *devide et impera* (разделяй и властвуй, лат.) и широко им применяется при разработке и проектировании любых сложных технических систем. Согласно первой части этого принципа, при проектировании сложной программной системы проводится ***алгоритмическая декомпозиция*** решаемой задачи.

Целью декомпозиции является представление разрабатываемой системы в виде взаимодействующих небольших подсистем (модулей или блоков), каждую из которых можно отладить (испытать) независимо от других. В этом случае при разработке системы, разделенной на «крупные» элементы или подсистемы, необходимо держать в уме информацию о гораздо меньшем числе деталей, чем в отсутствие такого разделения.

Наряду с термином *декомпозиция,* также используется термин ***структуризация*** проблемы, задачи или программы. Идеи разделения программ на относительно самостоятельные крупные части, реализующие определенные процедуры и функции и образующие определенную иерархию взаимосвязей, нашли отражение в ***структурном подходе*** к разработке и созданию программных средств. В программировании структурный подход появился с возникновением первых подпрограмм, процедур и функций, написанных в так называемом ***процедурно-ориентированном стиле.*** Данный стиль опирается на простое правило: определить переменные и константы, которые понадобится хранить в памяти компьютера, и описать или использовать алгоритм их обработки.

Теоретическое оформление структурный подход получил в конце 60-х — начале 70-х годов в работах Э. Дейкстры, А.П. Ершова, Э. Йодана, Н. Вирта, Э. Брукса и других теоретиков и практиков программирования. Следует отметить появление структурного программирования, в котором нашла определенное отражение идея упорядочивания структуры программы. ***Структурное программирование*** ориентирует на составление программ, структура которых близка к «дереву» операторов или блоков. Использование структуры типа «дерево» в качестве своеобразного эталона объясняется тем, что это простая для анализа и реализации структура.

Дальнейшее развитие структурного подхода привело к ***модульному программированию.*** Оно предусматривает декомпозицию прикладной задачи в виде иерархии взаимодействующих модулей или программ. Модуль, содержащий данные и процедуры их обработки, удобен для автономной разработки и создания. Специализация модулей по видам обработки и наличие в них данных определенных типов — это свойства, которые отражают генетическую связь модульного программирования и ООП.

Важнейшими инструментами производителей ПП, в которых находят отражение практически все аспекты эволюции, являются языки программирования. Язык программирования изначально ориентирован на компьютер и содержит набор типов данных, операторов, операций, функций и других операционных единиц языка, которые достаточно просто могут быть переведены в инструкции (команды) по управлению аппаратным и программным обеспечением компьютера. При этом желательно максимизировать эффективность трансляции предложений языка в машинные коды в смысле минимизации требуемой памяти, времени выполнения программы и стоимости создания транслятора. Вместе с тем язык программирования ориентирован на программиста и предоставляет средства для моделирования объектов, их свойств и поведения при решении прикладных задач в некоторой предметной области в виде программ.

Развитие языков в направлении повышения эффективности составления прикладных программ привело к разделению языков по следующим уровням.

1. Низкий уровень (машинно-ориентированные языки — языки ассемблера).
2. Высокий уровень (процедурно-ориентированные языки: FORTRAN, ALGOL, PL/1, Pascal).
3. Уровень решаемой задачи (проблемно-ориентированные языки — GPS, SQL).

Введение *типов данных* обозначило еще одно направление развития технологии программирования. Типизация данных предназначена как для облегчения составления программ, так и для автоматизации выявления ошибок использования данных в виде операндов и фактических параметров при вызове функций. Использование *структурных типов данных* позволяет, во-первых, упростить работу алгоритмиста при сопоставлении структур данных прикладной задачи и данных, обрабатываемых процедурами и функциями программных модулей, и, во-вторых, сократить объем рутинной работы программиста при кодировании алгоритма обработки.

Результатом обобщения понятия «тип данных» являются классы объектов (C++) или объектные типы (Pascal), которые могут содержать в качестве элементов не только данные определенного типа, но и методы их обработки (функции и процедуры).

Таким образом, по мере развития технологии программирования и в программах, и в типах данных все адекватнее отражалась структура решаемой прикладной задачи и осуществлялась соответствующая интеграция данных и программ в модулях. Одновременно с этим языки программирования пополнились средствами, необходимыми для описания подобных структур. Развитие идей абстрагирования и модульности привело к появлению в программировании объектного подхода.

Человек мыслит образами или объектами, он знает их свойства и манипулирует ими, сообразуясь с определенными событиями. Еще древним грекам принадлежит мысль о том, что мир можно рассматривать в виде объектов и событий. Рене ДеКарт' отмечал, что люди обычно имеют объектно-ориентированный взгляд на мир. Так, подумав о телефонном аппарате, человек может представить не только его форму и цвет, но и возможность позвонить, характер звучания звонка и ряд других свойств (в зависимости от его побуждений, технических знаний, фантазии).

Язык программирования позволяет описать свойства моделируемых объектов и порядок манипуляции с объектами или порядок их взаимодействия, сообразуясь с условиями решаемой задачи. Первые языки программирования ориентировались, с одной стороны, на математические объекты, а с другой — на определенную модель вычислителя (ЭВМ с архитектурой фон Неймана). Поэтому они содержали такие конструкции как переменная, константа, процедура, функция, формальные и фактические параметры. Программисты представляли свои программы в виде взаимодействующих подпрограмм (блоков, процедур, функций) и модулей. Характер программирования был процедурно-ориентированным, поскольку первостепенное внимание уделялось последовательностям действий с данными (процедурам). Соответственно такие языки программирования, как FORTRAN, ALGOL, COBOL, PL-1, C называют ***процедурно-ориентированными.***

Данные и подпрограммы объединялись в модули в соответствии с логикой проектировщиков, создающих сложные программные системы для определенных областей их применения. Логика интеграции в модули определялась рядом факторов, среди которых следует отметить свойства предметной области: данные и подпрограммы их обработки, соответствующие определенному классу объектов предметной области, объединялись в модуль. Например, модуль обработки строк содержал подпрограммы выполнения основных операций со строками: объединения, сравнения, копирования, нахождения вхождения подстроки в строку, вычисления длины строки.

Развитием идеи модульного программирования является сопоставление объектам предметной области (моделируемым объектам) программных конструкций, называемых ***объектами,*** *объектными типами* или ***классами*** (моделирующими объектами). Моделирующие объекты содержат данные и подпрограммы, которые описывают свойства моделируемых объектов. Так, данные могут отражать признаковые или количественные свойства (масса, длина, мощность, цена, наличие, видимость и т. п.), а подпрограммы отражают поведенческие или операционные свойства (изменить массу, вычислить мощность, установить цену, проверить наличие, сделать видимым и т. п.). Таким образом, при объектном подходе интеграция данных и процедур их обработки определяется структурой предметной области, т.е набором моделируемых объектов, их взаимосвязью или взаимодействием в рамках решаемой задачи.

Моделируемый объект всегда представляется человеку чем-то единым, целостным, хотя может состоять из частей или других объектов. Например, телефонный аппарат состоит из трубки, провода, корпуса с номеронабирателем, а трубка содержит микрофон, динамик и провода и т. д. Целостное представление объекта в виде взаимосвязанной совокупности его свойств или компонентов является базовым принципом объектного подхода.

Объектный подход начал развиваться в программировании со второй половины 60-х годов (язык Simula-67) и нашел свое отражение в языках Smalltalk, CLOS, Ada и в ряде других. Эти языки называются объектными. Иерархическая классификация объектов **и** наследование свойств являются отправными идеями появившегося в начале 80-х годов объектно-ориентированного подхода. Одной из причин сравнительно медленного становления объектно-ориентированного стиля программирования является его существенное отличие от господствовавшего процедурно-ориентированного стиля.

0.2. Концепции объектно-ориентированного программирования

ООП является третьим крупным этапом (после структурного и модульного программирования) в процессе развития структурного подхода. Создаваемые в середине 70-х годов большие программные системы продемонстрировали, что в рамках процедурно-ориентированного стиля использование структурного подхода не дает желаемого эффекта. По мере увеличения числа компонентов в создаваемых программных системах число ошибок, связанных с неправильным использованием процедур и некорректным учетом взаимосвязей между компонентами, стало нелинейно расти. Сроки ввода в эксплуатацию этих систем постоянно срывались. Уменьшить число подобных ошибок и упростить их обнаружение могла позволить алгоритмическая декомпозиция, ориентирующаяся на «естественные» элементы (компоненты или объекты) пространства решаемой задачи. В этом случае на этапе кодирования и отладки упрощалось сопоставление программистских конструкций с моделируемыми объектами.

Такую декомпозицию будем называть объектно-ориентированным анализом пространства решаемой задачи или предметной области. Для описания результатов объектно-ориентированного анализа и последующего программного синтеза необходимы адекватные языковые средства, которые построены на определенных принципах. Рассмотрим их.

Основным понятием ООП является *объект* или ***класс*** в C++, который можно рассматривать с двух позиций. Во-первых, с позиции предметной области: *класс* соответствует определенному характерному (типичному) объекту этой области. Во-вторых, с позиции технологии программирования, реализующей это соответствие: «класс» в ООП — это определенная программная структура, которая обладает тремя важнейшими свойствами:

- инкапсуляции;
- наследования;
- полиморфизма.

Эти свойства используются программистом, а обеспечиваются объектно-ориентированным языком программирования (транслятором, реализующим это язык). Они позволяют адекватно отражать структуру предметной области.

Некоторые авторы называют данные свойства объектов принципами ООП. Но, по-видимому, это слишком узкое понимание сущности ООП, поскольку в стороне остались отнюдь не детали:

- объектно-ориентированный анализ предметной области и парадигма ООП;
- создание и уничтожение объектов,
- принципы организации взаимодействия объектов.

Объекты и классы

Концепция объектов предназначена для моделирования (отображения) понятий предметной области в виде программных единиц, объединяющих в себе атрибуты и поведение (состояние и функционирование) соответствующих объектов (сущностей) предметной области.

Класс объектов представляет собой программную структуру, в которой данные и функции образуют единое целое и отражают свойства и поведение этого целого в рамках моделируемой предметной области. В отличие от модуля, в котором на состав данных и функций накладывается меньше смысловых ограничений, в объекте присутствуют только те данные и функции, которые необходимы для описания свойств и поведения объекта определенного типа.

Объекты выделяются в процессе анализа предметной области с использованием идей абстрагирования от несущественного и классификации родственных объектов. Результатом объектно-ориентированного анализа являются ***классы объектов,*** которые присутствуют или в перспективе могут присутствовать в пространстве решаемой задачи и образуют ***иерархии классов,*** представляемые в виде *деревьев наследования свойств.*

Например, на основе анализа авиационной техники можно выделить класс объектов *Самолет.* При этом мы абстрагировались от таких свойств как: форма крыльев, длина фюзеляжа, используемые материалы конструкций, расположение крыльев и т. п. К числу основных свойств класса объектов *Самолет* можно отнести: скорость полета, размах крыльев, тип двигателя, грузоподъемность, высота полета, функциональное назначение и т. д.

Для примера приведем также иерархию классов *Авиамодели,* которую можно представить в следующем виде:

Авиамодель:

- кордовые модели:
  - гоночные;
  - скоростные;
  - пилотажные;
  - воздушного боя;
  - копии;
- модели свободного полета:
  - двигательные:
    - —таймерные;
    - —радиоуправляемые;
    - —копии;
  - резиномоторные;
  - планерные.

Класс объектов характеризуется уникальным набором свойств и ему присваивается уникальное имя, как и любому типу данных. В качестве переменных программы используются ***объекты*** определенного класса. Создаваемые объекты, даже одного класса, могут отличаться значениями (степенью проявления) свойств и, конечно, должны отличаться именами.

Инкапсуляция свойств объектов

Инкапсуляция (дословно — «содержание в оболочке») представляет собой объединение и локализацию в рамках объекта, как единого целого, данных и функций, обрабатывающих эти данные. В совокупности они отражают свойства объекта.

В C++ данные класса и объекта называются ***элементами данных*** или **полями,** а функции —

методами или ***элементами-функциями.***

Доступ к полям и методам объекта осуществляется через имя объекта и соот ветствующие имена полей и методов при помощи операций выбора «.» и «->». Это позволяет в максимальной степени изолировать содержание объекта от внеш него окружения, т. е. ограничить и наглядно контролировать доступ к элементам объекта. В результате замена или модификация полей и методов, инкапсулиро ванных в объект, как правило, не влечет за собой плохо контролируемых послед ствий для программы в целом. При необходимости указания имени объекта в теле описания этого объекта в C++ используется зарезервированное слово ***this,*** которое в рамках объекта является специальным синонимом имени объекта — указателем на объект.

Отметим, что именование классов, элементов данных и методов имеет большое значение в ООП. Названия должны либо совпадать с названиями, использующимися в предметной области, либо ясно отражать смысл или назначение (функциональность) именуемого класса, поля или метода. При этом не следует бояться длинных имен — затраты на написание сторицей окупятся при отладке и сопровождении прот. е..е.ного продукта. Текст подобной программы становится понятным без особых комментариев. Дополнительным средством т. е.е. е.ения доступа к данным и методам является описание элементов классов с помощью спецификаторов ***private, protected*** **и** ***public,*** которые определяют три соответствующих уровня доступа к компонентам класса: собственный, защищенный и общедоступный.

Для расширения доступа к элементам данных, имеющим атрибуты *private* или *protected,* в классе можно реализовать с атрибутом *public* специальные методы доступа к собственным и защищенным элементам данных.

Гибкое разграничение доступа позволяет уменьшить нежелательные (бесконтрольные) искажения свойств объекта или несанкционированное использование свойств классов.

Хорошим стилем в ООП считается организация доступа к элементам данных с помощью функций или методов без использования оператора присваивания. Конечно, это положение не должно являться догмой, но случаи отхода от него должны быть хорошо обдуманы.

Наследование свойств

Наследование есть свойство классов порождать своих потомков и наследовать свойства (элементы данных и методы) своих родителей. Класс-потомок автоматически наследует от родителя *все элементы данных* и *методы,* а также может содержать новые элементы данных и методы и даже заменять (перекрывать, переопределять) методы родителя или модифицировать (дополнять) их. Наследование в ООП позволяет адекватно отражать *родственные отношения* объектов предметной области. Если класс B обладает всеми свойствами класса A и еще имеет дополнительные свойства, то класс A называется ***базовым*** (родительским), а класс B называется ***наследником*** класса A. В C++ возможно *одиночное* (с одним родителем) и *множественное* (с несколькими родителями) наследование.

Родственные отношения, или ***отношения включения свойств классов,*** могут отражаться не только с помощью наследования, но и путем инкапсуляции* в классе в качестве элементов данных других классов. Например, стек может быть построен, как наследник одного класса — «элемент стека». Возможно построение стека, как класса, содержащего в качестве элемента данных объект класса «элемент стека».

Свойство наследования упрощает модификацию свойств классов, обеспечивает ООП исключительную гибкость и сокращает затраты на написание новых классов на основе старых (родителей). Программист обычно определяет базовый класс, обладающий наиболее общими свойствами, а затем создает последовательность потомков, которые обладают своими специфическими свойствами. В результате получается иерархия наследования свойств классов.

Базовые классы, или корни подобных деревьев, обладают такими абстрактными свойствами, что часто непосредственно в программах не используются, а необходимы «всего лишь» для порождения требуемых классов. Правильный выбор корня обеспечивает удобство «выращивания» дерева, т. е. простоту развития библиотеки классов Применение правила ООП ***«наследуй и изменяй свойства объектов»*** хорошо согласуется с поэтапным подходом к разработке и созданию больших программных систем.

Примеры родственных классов: Координаты на экране -> Цветная точка -» Прямая -> Прямоугольник. Здесь направление стрелки указывает порядок наследования свойств классов.

Полиморфизм поведенческих свойств объектов

Полиморфизм часто определяют как свойство объектов-родственников по-разному осуществлять однотипные (и даже одинаково поименованные) действия, т. е. однотипные действия у множества родственных классов имеют множество различных форм. Например, метод «нарисовать на экране» должен по-разному реализовываться для родственных классов «точка», «прямая», «ломанная», «прямоугольник». В ООП

действия или поведение объекта определяется набором методов. Изменяя алгоритм метода в потомках класса, программист придает наследникам специфические поведенческие свойства. Для изменения метода необходимо перекрыть его в потомке, т. е. объявить в потомке одноименный метод и реализовать в нем нужные действия, отражающие специфику потомка.

Свойство полиморфизма реализуется не только в механизме замещения (перекрытия) одноименных методов при наследовании свойств, но и в механизме виртуализации методов, или позднем связывании методов. Если замещение метода реализуется на этапе компиляции ***(раннее связывание*** объекта с методом), то замещет. е. е.етт. е.объявленного в описании класса виртуальным (virtual) происходит на этапе выполнения *(позднее связывание)*. Но ничего не дается даром: виртуализация требует дополнительных ресурсов памяти и времени. Виртуальные методы выполняются медленнее, из-за необходимости дополнительных действий по связыванию.

Создание и уничтожение объектов

Описание класса в ООП есть некоторая программная структура, которую используют при создании объектов, отличающихся именами и отдельными свойствами. Создание и удаление объектов осуществляется с помощью специальных методов, которые называются *конструкторами (constructor)* и *деструкторами (destructor)* соответственно.

Конструктор класса создает и инициализирует объекты, а также может выполнять подготовку механизма позднего связывания, необходимого для использования виртуальных функций. *Деструктор* класса выполняет действия, завершающие работу с объектом, например: освобождает динамическую память, восстанавливает экран, закрывает файловые переменные.

Объекты можно размещать и в динамической памяти. Для этого необходимы указатели на эти объекты.

Взаимодействие объектов и сообщения

Отношение родства или включения свойств — это только один из многих типов отношений между классами. Еще одним важнейшим типом отношений между классами и между объектами являются отношения взаимодействия или отношения «клиент-сервер». Отношения родства также подразумевают определенное отношение взаимодействия, ограниченное свойством инкапсуляции.

Благодаря инкапсуляции объекты так хорошо изолируются друг от друга, что необходимо специально заботится о механизме их взаимодействия в программе. Это касается, в первую очередь, независимых объектов, а не тех, которые погружены в другие объекты в виде элементов данных этих объектов. Но и в последнем случае взаимодействие может быть ограниченно синтаксисом языка и соглашениями о видимости (доступности) элементов данных и методов родственных объектов и инкапсулированных объектов. Поэтому для его расширения требуется специально заботиться об этом.

Часто связь устанавливают при помощи *указателей,* что делает программу похожей на структуру данных в динамической памяти. Примерами такого способа являются программы, написанные с помощью библиотек Turbo Vision и Object Windows. Основным содержанием программы являются описание классов и совокупности взаимодействующих объектов. В библиотеках классов для доступа объекта к самому себе (внутри себя) используется зарезервированное слово *this,* являющееся синонимом имени объекта, в контексте которого оно используется.

Вторым способом обмена информацией между объектами является передача через *глобальную переменную* (буфер сообщений), которой один объект присваивает значение, а другой — считывает. Данный способ прост в реализации, но требует тщательного контроля обмена сообщениями со стороны программиста, поскольку структура взаимодействия более скрыта от программиста, чем при первом способе.

В ООП термин «послать объекту сообщение» часто трактуют, как вызов метода' объекта, которому посылается сообщение, определяемое данным вызовом.

0.3. Этапы объектно-ориентированного программирования

Программа, решающая некоторую задачу, заключает в себе описание части мира, относящегося к этой задаче, или определенной предметной области. Описание действительности в форме взаимодействующих объектов, по-видимому, естественнее, чем в форме иерархии подпрограмм, и поэтому облегчает программное моделирование в некоторой предметной области.

В процессе программирования в объектно-ориентированном стиле можно выделить следующие этапы:

1. Определение основных понятий предметной области и соответствующих им

классов, имеющих определенные свойства (возможные состояния и действия). Обо
снование вариантов создания объектов.

2. Определение или формулирование принципов взаимодействия классов и взаи
модействия объектов в рамках программной системы.

3. Установление иерархии взаимосвязи свойств родственных классов.

4. Реализация иерархии классов с помощью механизмов инкапсуляции, наследо
вания и полиморфизма.

5. Для каждого класса реализация полного набора методов для управления свой
ствами объектов.

В результате будет создана объектно-ориентированная среда, или библиотека классов, позволяющая решать задачи моделирования в определенной предметной области.

Первые три этапа и являются объектно-ориентированным анализом предметной области.

В заключение раздела отметим, что ООП обладает следующими достоинствами:

• использование более естественных понятий и имен из повседневной жизни или предметной области;

• простота введения новых понятий на основе старых;

• отображение в библиотеке классов наиболее общих свойств и отношений между объектами моделируемой предметной области;

• естественность отображения пространства решаемой задачи в пространство объектов программы;

• простота внесения изменений в классы, объекты и программу в целом;

• полиморфизм классов упрощает составление и понимание программ;

• локализация свойств и поведения на основе инкапсуляции и разграничения дос
тупа упрощают понимание структуры программы и ее отладку;

• передача параметров в конструкторах объектов повышает гибкость создаваемых программных систем.

В качестве недостатков ООП можно отметить следующие:

• снижение быстродействия программ, особенно при использовании виртуальных методов;

• большие затраты времени на разработку библиотеки классов, поэтому ООП целесообразнот. Использовать при производстве ПО или при создании больших программ, а не при написании маленьких единичных программ;

• необходимость анализа полной иерархии классов для правильного использования свойств базовых классов.

Целесообразность применения технологии ООП для создания конкретной программной системы определяется двумя главными факторами:

• спецификой предметной области и решаемой прикладной задачи;_

• особенностями используемого языка программирования (транслятора).

Предметная область может быть в большей или меньшей степени разработанной для использования технологии ООП. В разработанных областях ясны классы, их иерархия и отношения взаимодействия. В неразработанных областях использование ООП предусматривает гораздо большие затраты на первоначальных этапах разработки (объектно-ориентированный анализ, проектирование и создание библиотеки классов).

Известны области, в которых требуется максимальная эффективность программного кода в смысле быстродействия и затрат памяти (системы реального времени, сверхминиатюрные средства). В таких областях применение ООП может быть оправдано или снижением стоимости разработки ПП, или удовлетворительным качеством генерируемого программного кода.

Контрольные вопросы и задания

1. Назовите основные черты эволюции структурного подхода в программировании.

2. Укажите важнейшие факторы, влияющие на эволюцию методов проектирования и создания программных продуктов.

3. Каковы различия модульного и объектно-ориентированного программирования?

4. Укажите основной принцип понижения сложности программ.

5. Назовите основные различия процедурно-ориентированного и объектно-ориентированного стилей программирования.

6. Укажите, для чего предназначена типизация данных. Что дает использование структурных типов данных?

7. Перечислите этапы разработки программ в объектно-ориентированном стиле.

8. Охарактеризуйте свойства инкапсуляции.

9. Каким образом осуществляется доступ к элементам данных и методам класса или объекта?

10. Каким образом устанавливаются дополнительные ограничения к данным и методам классов?
11. Каково назначение зарезервированного слова *this?*
12. Как соотносятся между собой и что означают понятия *класс объекта* и *объект?*
13. Дайте определение полиморфизма.
14. Назовите достоинства применения ООП.
15. Какими факторами определяется целесообразность применения технологии ООП?
16. С помощью каких методов выполняется создание и удаление объектов?
17. Назовите способы установления связи и обмена информацией между объектами.
18. В чем суть наследования? Какие разновидности наследования имеются в языках программирования?
19. Предложите вариант иерархии классов в области транспортных средств.
20. Предложите вариант иерархии классов в области вычислительной техники.
21. Предложите вариант иерархии классов в животном мире.