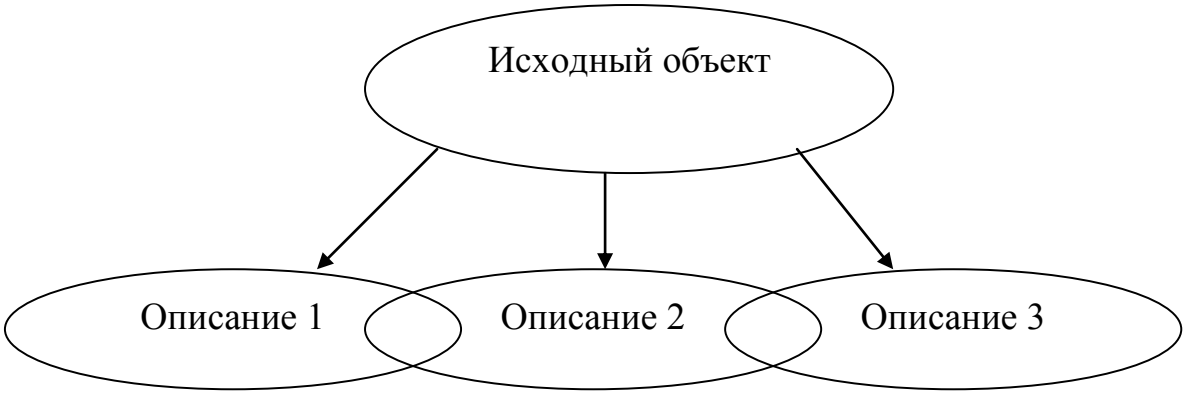
1. Понятие об объектах. Принцип инкапсуляции

Объект – это достаточно общее и широкое понятие, которое может иметь разные трактовки. В широком смысле слова **объект – это любая сущность, имеющая некоторый набор свойств (параметров, характеристик) и обладающая некоторым поведением (функциональностью).**

Отсюда следует, что объектом может быть практически все, что угодно – **материальные** предметы (компьютер, автомобиль, человек), **логические** понятия (файл, документ, таблица, список, управляющая кнопка, окно в многооконной графической системе, форма, приложение), **математические** понятия (например, геометрические объекты типа отрезок или окружность).

Под объектом в узком смысле можно понимать некоторое **формализованное описание** рассматриваемой сущности, т.е. **модель** исходного объекта**.** Для такого описания нужен некоторый язык, например – язык программирования. В этом случае объект становится элементом языка, и именно в таком контексте он и будет рассматриваться далее.

Моделирование реальных объектов – это важнейший этап разработки объектных программ, во многом определяющий успех всего проекта. Необходимо понимать, что любая модель – это только **часть** исходного моделируемого объекта, отражающая те свойства и атрибуты объекта, которые **наиболее важны** с точки зрения решаемой задачи. Изменение постановки задачи может привести к существенному изменению модели. Особенно это характерно для сложных объектов. Вряд ли имеет смысл строить всеобъемлющую модель “на все случаи жизни”, поскольку такая модель будет очень громоздкой и практически непригодной для использования. Поэтому один из важнейших принципов построения моделей – это принцип **абстрагирования**, т.е. выделение наиболее существенных для решаемой задачи черт исходного объекта и отбрасывание второстепенных деталей. Отсюда следует, что для одного и того же исходного объекта можно построить **разные** модели, каждая из которых будет отражать только наиболее важные в данный момент черты объекта.



Например, для исходного объекта “человек” можно построить следующие информационные модели:

- человек как студент учебного заведения
- человек как сотрудник организации
- человек как пациент учреждений здравоохранения
- человек как клиент государственных органов управления и т.д.

Из этого примера видно, что весьма часто различные модели одной и той же исходной сущности «пересекаются», имеют общие свойства (это отражено и на приведенной выше схеме): все студенты, сотрудники, клиенты имеют фамилию, дату рождения, место проживания, пол и т.д.

Как показала практика последних 10-15 лет, использование объектов при разработке **сложных** информационных систем оказалось очень удачным решением. Заказчики информационных систем, как правило, не являются специалистами в области программного обеспечения и поэтому не воспринимают (и не должны!) такие программистские термины как переменные, подпрограммы, массивы, файлы и т.д. Гораздо легче воспринимаются **объекты** как элементы предметной области.

Разработчики на основе общения с заказчиком и на основе анализа предметной области должны построить **адекватные описания основных сущностей** решаемой задачи, перевести эти описания в необходимый **формальный** вид и создать объектную программу как **набор взаимодействующих объектов.**

Основные этапы разработки объектных программ укрупненно можно представить следующим образом:

1. **Абстрагирование**: создание **неформального** описания объектов

2. **Формализация**: создание формальных описаний (**моделей**) объектов

3.**Реализация**: разработка объектных программ как наборов взаимодействующих объектов

Достоинством объектных программ является их четкая структуризация – каждый объект выполняет свою строго определенную задачу, а их совместная работа позволяет достичь поставленных целей.

Программный объект – это условное понятие, с которым связывается набор некоторых данных и программный код обработки этих данных

объект : данные + программный код

Объединение в рамках объекта некоторых данных и соответствующего программного кода рассматривается как проявление одного из базовых принципов объектного подхода – принципа **инкапсуляции (encapsulation).**

Инкапсуляция позволяет рассматривать объект в виде некоторой достаточно самостоятельной **программной единицы**, полностью отвечающей за хранение и обработку своих данных и предоставляющей посторонним пользователям четко определенный набор услуг. Однако объект это не только поставщик услуг, но чаще всего еще и потребитель услуг других объектов.

Относительно связываемых с объектом **данных** надо отметить следующее:

- каждый элемент данных часто называют **свойством** объекта (хотя есть и различия в трактовке этого термина)
- объект может содержать **любое разумное** число данных-свойств

- набор данных-свойств определяется при описании объекта и при выполнении программы изменяться не может, могут изменяться лишь **значения** свойств
- текущие **значения** свойств определяют текущее **состояние** объекта
- для хранения значений свойств при выполнении программы необходима **память** (как для обычных переменных), и поэтому объекты программы **потребляют** оперативную память
- свойства могут иметь разные типы: **простейшие** (целочисленные, символьные, логические), **структурные** (строки, массивы, списки) и даже **объектные** (этот очень важный случай рассматривается в разделе «Взаимодействие объектов: агрегация и композиция»)
- в соответствии с принципом **инкапсуляции** элементы данных **рекомендуется** делать **недоступными** для прямого использования за пределами объекта (**закрытость** данных)

В качестве **примеров** свойств можно привести:

- объект «Студент»: фамилия, имя, дата рождения, место проживания, группа, специальность, массив оценок;
- объект «Файл»: имя, размер, дата создания, тип;
- объект «Окно»: положение на экране, размеры, тип, фон заполнения, тип рамки, оформление заголовка;
- объект «Автомобиль»: стоимость, тип, марка, мощность двигателя, цвет и т.д.;
- объект «Окружность»: координаты центра, радиус, цвет.

Относительно связываемого с объектом **программного кода** необходимо отметить следующие моменты:

- программный код разбивается на отдельные **подпрограммы**, которые принято называть **методами**

- набор методов определяет выполняемые объектом функции и тем самым реализует поведение объекта
- набор методов определяется при описании объекта и при выполнении программы уже не изменяется
- методы могут иметь **ограничения по доступности**: некоторые методы можно сделать недоступными (**закрытыми**) за пределами объекта, но всегда должен быть определен набор **открытых** методов, образующих внешний интерфейс объекта
- вызов одного из открытых методов соответствует запросу услуги, реализуемой данным методом
- среди методов выделяют один или несколько специальных **методов-конструкторов** и иногда – один **метод-деструктор**, назначение которых рассматривается чуть дальше
- в соответствии с принципом **инкапсуляции** для доступа извне к закрытым данным могут вводиться специальные **методы доступа**

Учитывая огромную важность методов-конструкторов, рассмотрим отдельно их назначение и особенности использования.

Конструктор отвечает за **динамическое** создание нового объекта при **выполнении** программы, которое включает в себя:

- **выделение памяти**, необходимой для хранения значений свойств создаваемого объекта (совместно с операционной системой и средой поддержки выполнения программ)
- **занесение** в выделенную область **начальных значений** свойств создаваемого объекта

Важно понимать, что пока объект не создан, он не может предоставлять никакие услуги и поэтому конструктор должен вызываться **раньше** всех остальных методов. Важность конструктора определяется тем, что он **гарантированно** создает объект с какими-то **начальными** значениями

свойств. Это позволяет устранить целый класс неприятных логических ошибок, таких как использование неинициализированных данных.

Для одного и того же объекта можно предусмотреть **несколько** различных конструкторов, которые **по-разному** инициализируют свойства создаваемого объекта. Начальные значения свойств часто передаются конструктору как входные параметры.

Деструктор отвечает за уничтожение объекта, т.е. освобождение памяти, выделенной объекту. В отличие от конструкторов, механизм деструкторов реализован не во всех языках: деструкторы есть в языке C++, в языке Delphi Pascal, предусмотрены (но практически не используются) в языке C# и отсутствуют в языке Java. В последних двух языках вместо деструкторов реализован специальный механизм «**сборки мусора**» (garbage collect), который способен при необходимости выполнять автоматическое освобождение памяти.

Еще одна группа очень часто используемых методов – это **методы доступа** к закрытым свойствам объекта. Введение таких методов позволяет организовать **контролируемый** доступ к внутренним данным объекта. В общем случае для каждого закрытого свойства можно:

- ввести **два** метода доступа: для **чтения** хранящегося в свойстве значения (часто такие методы называют **get-методами**) и для **изменения** значения свойства (**set-метод**)
- ввести только **один get**-метод: свойство доступно только для **чтения**
- не вводить **ни одного** метода доступа, что соответствует **полной** закрытости элемента данных

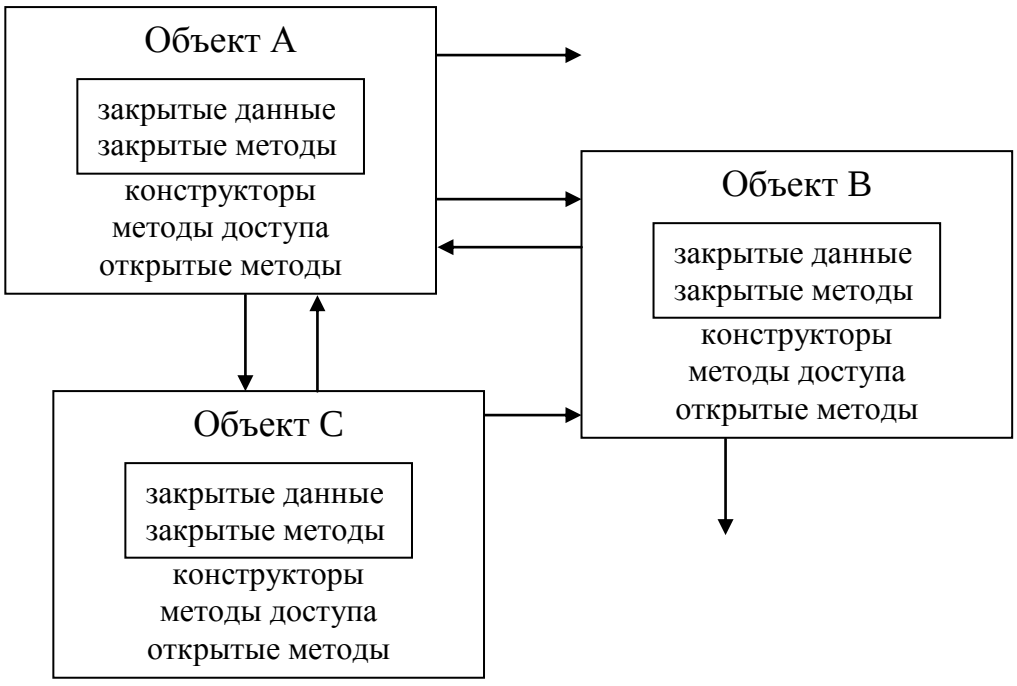
Набор используемых с каждым свойством методов доступа определяется при разработке объекта и позволяет для каждого свойства организовать **необходимый уровень доступа.**

Иногда set-методы кроме изменения значений свойств выполняют некоторую **дополнительную** работу, например – проверяют новые значения свойств.

Кроме конструкторов и методов доступа, объекты практически всегда имеют еще и некоторый набор **специфических** методов, определяющих **функциональность** объекта. Например:

- объект «Окружность»: отображение на экране, перемещение, растяжение/сжатие, вычисление длины окружности.
- объект «Список»: добавление элемента, удаление элемента, поиск элемента, сортировка элементов;
- объект «Окно»: отображение, перемещение, изменение атрибутов, изменение размеров;
- объект «Студент»: посещение занятий, выполнение заданий, сдача контрольных точек, оплата обучения;

В итоге объектная программа представляет собой **набор взаимодействующих объектов,** которые обращаются друг к другу за выполнением необходимых действий.



При этом каждый объект проходит определенный **жизненный цикл:**

- объект **создается** методом-конструктором

- объект **используется** другими объектами, предоставляя им свои открытые методы и неявно – закрытые данные
- объект **уничтожается** (явно деструктором или неявно механизмом сборки мусора)

Очевидно, что при работе объектной программы одновременно может существовать **множество** однотипных программных объектов (например - множество файлов, множество окон, множество студентов). Поэтому необходим инструмент **формального описания** таких **однотипных объектов** и в качестве такого инструмента выступает следующее важнейшее понятие объектного подхода - **класс.**