Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
| Институт информационных технологий и анализа данных |
| наименование института  Кафедра автоматизированных систем |
| наименование кафедры |

|  |  |  |
| --- | --- | --- |
| Допускаю к защите | | |
| Руководитель |  |  |
|  |  | Е. А. Осипова |
|  | И.О. Фамилия |

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ HTML ДОКУМЕНТА

наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

ИНФОРМАТИКА

1.003.00.00 ПЗ

обозначение документа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | ИСТб 19-2 |  |  |  | Солопов Д. Д. |
|  | Шифр |  | Подпись |  | И.О. Фамилия |
| Нормоконтроль |  |  |  |  | Е. А. Осипова |
|  |  |  | Подпись |  | И.О. Фамилия |
|  |  |  |  |  |  |
| Курсовая работа защищена с оценкой | |  | | | |

Иркутск 2019 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ   
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

|  |  |  |  |
| --- | --- | --- | --- |
| **По курсу** | Информатика | | |
| **Студенту** | Солопову Д. Д. | | |
|  | (фамилия, инициалы) | | |
| **Тема работы:** | | Введение в объектно-ориентированное программирование | |
| **Исходные данные:** | | | Разработать сайт на тему «Введение в объектно-ориентированное программирование» |
| 1 Курсовая работа «Проектирование и реализация HTML документа» | | | |
| [Электронный ресурс]: руководство и методические указания для студентов | | | |
| /ИРНИТУ; сост. З.А. Бахвалова, И.А. Серышева. – Иркутск: ИРНИТУ, | | | |
| 2017. – 43 с. URL: <http://elib.istu.edu/viewer/view.php?file=/files/er-1221.pdf> | | | |
| 2 С.В. Симонович и др. Информатика: Базовый курс. СПб.: Питер, 2010. – 640 с. | | | |
| 3 Панфилов К. Создание веб-сайта от замысла до реализации [Электронный | | | |
| ресурс]: учебное пособие /К. Панфилов. – М.: ДМК Пресс, 2009. – 440 с.  URL: <http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=1072> | | | |
| 4 Сухов К. HTML5 – путеводитель по технологии [Электронный ресурс]: | | | |
| /К. Сухов. – М.: ДМК Пресс, 2012. – 312 с. | | | |
| URL:<http://e.lanbook.com/books/element.php?pl1_cid=25&pl1_id=4816> | | | |
| 5 Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и | | | |
| устройств. СПб.: Питер, 2014. – 298 с. | | | |
| 6 Чак Муссиано и Билл Кеннеди. HTML и XHTML. Подробное руководство. | | | |
| 6-е издание. СПб.: Символ-Плюс, 2008 г. | | | |
| 7 Роберт Агулар. HTML и CSS. Основа любого сайта. М.: Эксмо, 2010 г. – 319 с. | | | |
| 8 Дронов В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web- | | | |
| сайтов. СПб.: БХВ, 2011 г. – 414 с. | | | |
| 9 Теги HTML и справочник СSS. [Сайт] URL: <http://htmlbook.ru/> | | | |

Графическая часть на \_\_\_\_\_\_\_\_\_\_\_\_\_\_ листах.

Дата выдачи задания “ ” 2019г.

|  |  |  |  |
| --- | --- | --- | --- |
| Задание получил |  |  | Д. Д. Солопов |
|  | подпись |  | И.О.Фамилия |

Дата представления работы руководителю “ ” 2019г.

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель курсовой работы |  |  | Е. А. Осипова |
|  | подпись |  | И.О.Фамилия |

**Содержание**

[Введение. 4](#_Toc27986157)

[1.Задание курсовой работы: 5](#_Toc27986158)

[2.Проектная часть 5](#_Toc27986159)

[2.1 Логическая структура сайта: 5](#_Toc27986160)

[2.2 Физическая структура сайта 5](#_Toc27986161)

[2.3 Исходный текст шаблонов сайта 7](#_Toc27986162)

[2.4 Тестирование сайта: 27](#_Toc27986168)

[2.5 Порядок разработки сайта 30](#_Toc27986169)

[2.6 Использованное ПО 31](#_Toc27986170)

[Список используемых источников 32](#_Toc27986171)

[Приложение А. Содержание сайта 33](#_Toc27986172)

Введение.

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

CSS (англ. Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

**Цель курсовой работы:**

Получение навыков проектирования и разработки HTML документа с использованием CSS, редакторов кода и редакторов изображений.

**Что необходимо освоить и изучить:**

* гипертекстовый язык разметки документа HTML 5;
* организацию структуры сайта;
* принципы построения системы навигации;
* основы дизайна сайта, понятие шаблона сайта;
* основы CSS – каскадных таблиц стилей;
* вёрстку страниц с использованием таблиц и фреймов;
* типовые элементы веб-страницы;
* оптимизация изображений и фоновых рисунков;
* внедрение скриптов в html-документы;
* использование активных изображений;
* тестирование и отладка готового кода;
* особенности популярных браузеров и как они работают с тегами и стилями.

1.Задание курсовой работы:

Создать web-сайт на тему: «Введение в объектно-ориентированное программирование».

2.Проектная часть

2.1 Логическая структура сайта:

Логическая структура сайта имеет решётчатую структуру, так как все страницы взаимосвязаны между собой, а это означает что пользователь может перемещаться между страницами различным способом (см. рисунок 1)

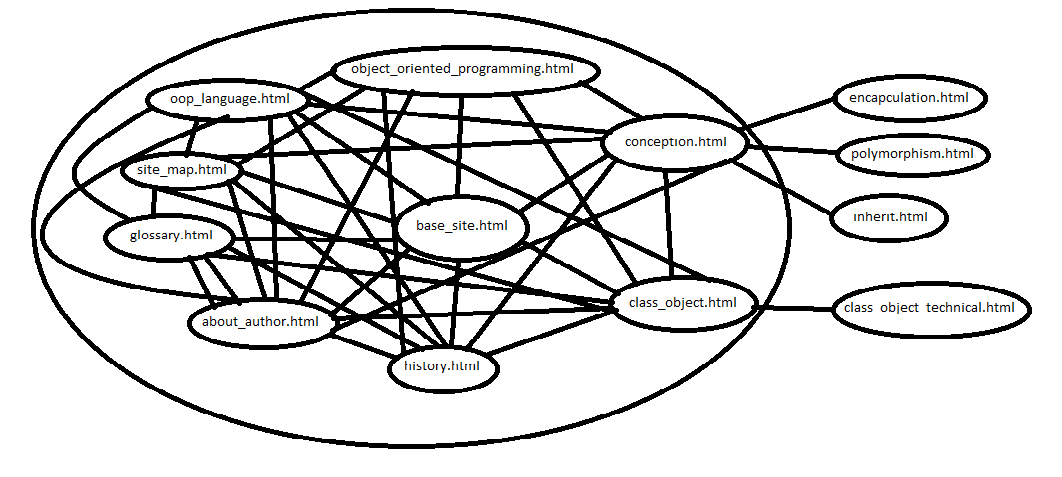


Рисунок 1 – Логическая структура сайта

2.2 Физическая структура сайта

В таблице приведена структура папки, в которую вложены все HTML-страницы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Папка | Имя файла | Принадлежность к html-странице | Размер(Кб) |
| 1 | Курсовая работа/ | base\_site.html | Главная страница сайта с использованием фрейма | 5 |
| 2 | Курсовая работа/ | history.html | История объектно-ориентированного программирования | 10 |
| 3 | Курсовая работа/ | object\_oriented\_programming.html | Объектно-ориентированное программирование | 26 |
| 4 | Курсовая работа/ | class\_object.html | Классы и объекты | 25 |
| 5 | Курсовая работа/ | class\_object\_technical.html | Классы и объекты. Конструкции языка | 18 |
| 6 | Курсовая работа/ | conception.html | Концепции объектно-ориентированного программирования | 34 |
| 7 | Курсовая работа/ | inherit.html | Наследование | 10 |
| 8 | Курсовая работа/ | encapculation.html | Инкапсуляция | 14 |
| 9 | Курсовая работа/ | polymorphism.html | Полиморфизм | 11 |
| 10 | Курсовая работа/ | oop\_language.html | Объектно-ориентированные языки | 11 |
| 11 | Курсовая работа/ | glossary.html | Глоссарий | 13 |
| 12 | Курсовая работа/ | site\_map.html | Карта сайта | 6 |
| 13 | Курсовая работа/ | about\_author.html | Об авторе | 6 |
| 14 | Курсовая работа/ | DataCSS.css | Каскадная таблица стилей | 3 |
| 15 | Курсовая работа/ | left\_point.png | Картинка | 5 |
| 16 | Курсовая работа/ | right\_point.png | Картинка | 5 |
| 17 | Курсовая работа/ | up\_point.png | Картинка | 5 |
| 17 | Курсовая работа/ | vk.jpg | Картинка | 17 |

2.3 Исходный текст шаблонов сайта

С использованием фрейма, таблиц и блоков (base\_site.html)

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link rel="stylesheet" href="DataCSS.css" type="text/css">

</head>

<body>

<div class="very\_top\_block"> <br>Введение в объектно-ориентированное программирование </div>

<div style="margin: 0px; width: 100%"></div>

<table class="class\_top\_table">

<tbody>

<tr>

<td style="width: 25%;">

Главная страница

</td>

<td style="width: 25%;">

<a href="glossary.html"> Глоссарий </a>

</td>

<td style="width: 25%;">

<a href="site\_map.html"> Карта сайта </a>

</td>

<td style="width: 25%;">

<a href="about\_author.html"> Об авторе </a>

</td>

</tr>

</tbody>

</table>

</div>

<div style="margin:0px; width: 100%">

<table class="class\_base\_table">

<tbody>

<tr>

<td class="class\_cell\_left">

<ul class="class\_ul">

<li> <a href="history.html"> История объектно-ориентированного программирования </a> </li><br>

<li> <a href="object\_oriented\_programming.html"> Объектно-ориентированное программирование </a> </li><br>

<li> <a href="class\_object.html"> Классы и объекты </a></li><br>

<li> <a href="conception.html"> Концепции объектно-ориентированного программирования </a> </li><br>

<li> <a href="oop\_language.html"> Объектно-ориентированные языки </a></li><br>

</ul>

</td>

<td class="class\_cell\_right">

<center>

<h1> О сайте </h1>

</center>

Сайт содержит информацию об объектно-ориентированном программировании.

Представлена информация об истории, основных концепциях и понятиях ООП, а также о языках,

которые поддерживают ООП.

<br><br>

Для практической реализации концепций и понятий ООП, автор использует язык C++. <br><br>

<iframe width="853" height="480" src="https://www.youtube.com/embed/zqxVI\_kEdq8" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

<br><br>

</td>

</tr>

</tbody>

</table>

</div>

<div>

<table class="class\_table\_down" cellspacing="0">

<tbody>

<td class="class\_cell\_down">

<br>Контактная информация: <a href="https://vk.com/id560798514"> <img src="vk.jpg" height="35px" width="40px" alt="Страница в ВКонтакте" style="vertical-align: middle"> </a>

<br><br>

</td>

<td class="class\_cell\_down">

<br>

<right>

<a href="base\_site.html"> <img src="up\_point.png" height="35px" width="40px"> </a>

</right>

</td>

</tbody>

</table>

</div>

</body>

</html>

Исходный текст шаблона сайта с использованием таблиц(class\_object.html)

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link rel="stylesheet" href="DataCSS.css" type="text/css">

</head>

<body>

<div class="very\_top\_block"> <br>Введение в объектно-ориентированное программирование </div>

<div style="margin: 0px; width: 100%"></div>

<table class="class\_top\_table">

<tbody>

<tr>

<td style="width: 25%;">

<a href="base\_site.html"> Главная страница </a>

</td>

<td style="width: 25%;">

<a href="glossary.html"> Глоссарий </a>

</td>

<td style="width: 25%;">

<a href="site\_map.html"> Карта сайта </a>

</td>

<td style="width: 25%;">

<a href="about\_author.html"> Об авторе </a>

</td>

</tr>

</tbody>

</table>

</div>

<div style="margin:0px; width: 100%">

<table class="class\_base\_table">

<tbody>

<tr>

<td class="class\_cell\_left">

<ul class="class\_ul">

<li> <a href="history.html"> История объектно-ориентированного программирования </a> </li><br>

<li> <a href="object\_oriented\_programming.html"> Объектно-ориентированное программирование </a> </li><br>

<li> Классы и объекты <br><br>

<div style="padding-left: 10px;">

<ul class="class\_ul\_ul">

<li> <a href="class\_object.html"> Понятие класса и объекта</a></li><br>

<li> Класс и объект. Конструкции языка </li><br>

</ul>

</div>

</li><br>

<li> <a href="conception.html"> Концепции объектно-ориентированного программирования </a> </li><br>

<li> <a href="oop\_language.html"> Объектно-ориентированные языки </a></li><br>

</ul>

</td>

<td class="class\_cell\_right">

<div>

<table class="class\_table\_down">

<tbody>

<tr>

<td style="text-align: left;">

<a href="class\_object.html"> <img src="left\_point.png" width="35" height="40" alt="Переход влево"> </a>

</td>

<td style="text-align: right;">

<a href="conception.html"> <img src="right\_point.png" width="35" height="40" alt="Переход вправо"> </a>

</td>

</tr>

</tbody>

</table>

</div>

<center>

<h1> Класс и объект. Конструкции языка </h1>

</center>

<br>

В предыдущей статье была дана теоретическая информация относительно объектов и классов. В этой статье будет дано

описание объектов и классов с точки зрения конструкций языка ну и практики как таковой.

<br><br>

<em> Класс </em>

<br><br>

В синтаксис определения класса входит ключевое слово <i>class</i>, имя класса и блок кода, в котором определяются

поля и методы.

<br><br>

<i> Синтаксис определение класса: </i><br>

<div style="padding-left: 15px;">

<code>

class name\_class{<br>

//блок кода<br>

}<br>

</code>

</div>

<br><br>

<i> Объявление объекта класса </i> в блоке кода любой функции будет выглядеть следующим образом:<br>

<div style="padding-left: 15px;">

<code>

name\_class object;

</code>

</div>

<br><br>

Как видно из синтаксиса объявления объекта класса name\_class object, синтаксис очень похож на объявление переменных.

Здесь стоит обратить внимание на определение класса из википедии. Класс очень схож с типом данных, и отсюда явно видна аналогия

в синтаксисе между объявлением простой переменной и объявлением объекта(переменной сложного типа).<br><br>

<em> Спецификаторы доступа </em><br><br>

В блоке кода определения класса методы и поля разделяются на три секции: <i> public, private, protected </i><br>

Зачем вообще такое разделение? Каждый объект класса имеет возможность обращаться к определённым полям и методам.

Спецификаторы доступа управляют доступом объекта к полям и методам, хранящимся в определённой секции.<br>

В секцию входит спецификатор, множество состоящее из операторов объявления полей и объявления/определения методов,

и сама секция идёт пока не встретит завершения блока кода определения класса или другой спецификатор.<br><br>

<i> Пример: </i>

<div style="padding-left: 15px">

<code>

class CLASS{<br>

&nbsp;&nbsp;public:<br>

&nbsp;&nbsp;&nbsp; //начало секции public<br>

&nbsp;&nbsp;&nbsp; int a;<br>

&nbsp;&nbsp;&nbsp; int b;<br>

&nbsp;&nbsp;&nbsp; //конец секции public

&nbsp;&nbsp;private:<br>

&nbsp;&nbsp;&nbsp; //начало секции private<br>

&nbsp;&nbsp;&nbsp; double a;<br>

&nbsp;&nbsp;&nbsp; double c;<br>

&nbsp;&nbsp;&nbsp; //конец секции private<br>

}<br><br>

</code>

</div>

В примере сделана попытка показать деление данных блока кода класса на секции. Сейчас мы рассмотрим описание всех

спецификаторов доступа. То есть, если спецификаторы доступа ограничивают объекту доступ к данным находящимся в определённой

секции, то каким образом они это делают? Сейчас мы и выясним.<br><br>

<em>public</em><br><br>

Всё что находится в секции public является доступным объекту класса через операцию точки(.) или косвенного обращения(->).

Т.е. объект класса имеет доступ к любым полям и методам класса, объектом которого он является, в котором эти поля и методы

определены/объявлены.<br>

<i> Пример: </i>

<div style="padding-left: 15px">

<code>

class CLASS{<br>

&nbsp;&nbsp;public:<br>

&nbsp;&nbsp;&nbsp;//начало секции public<br>

&nbsp;&nbsp;&nbsp;int a;<br>

&nbsp;&nbsp;&nbsp;double b;<br>

&nbsp;&nbsp;&nbsp;//конец секции private<br>

};<br><br>

void function(){<br>

&nbsp;&nbsp;CLASS object;<br>

&nbsp;&nbsp;object.a = 90;<br>

&nbsp;&nbsp;object.b = 120.312<br>

}<br><br><br>

</code>

</div>

Как видно из примера, объект класса CLASS object, без проблем имеет доступ к своим полям a и b, и изменяет их значения.

Объект класса CLASS object обращается к полям и методам через операцию точки(.), или, в случае если объект - указатель, с помощью

операции косвенного обращения(->).

<br><br>

<em>private</em>

<br><br>

Данный спецификатор ограничивает объекту класса доступ к полям и методам полностью. Т.е. объект класса не может обращаться к полями

и методам класса, которые находятся в секции private. Полное ограничение.<br>

<i> Пример: </i>

<div style="padding-left: 15px">

<code>

class CLASS{<br>

&nbsp;&nbsp;private:<br>

&nbsp;&nbsp;&nbsp;//начало секции private<br>

&nbsp;&nbsp;&nbsp;int a;<br>

&nbsp;&nbsp;&nbsp;double b;<br>

&nbsp;&nbsp;&nbsp;//конец секции private<br>

};<br><br>

void function(){<br>

&nbsp;&nbsp;CLASS object;<br>

&nbsp;&nbsp;object.a = 90;//ERROR<br>

&nbsp;&nbsp;object.b = 120.312//ERROR<br>

}<br><br><br>

</code>

</div>

В примере видно, что при обращении к полям класса CLASS, объект object обращается к полям которые находятся в секции

private, и результат таких действий - ошибка.

<br><br>

<em>protected</em>

<br><br>

Данный спецификатор для обычного объекта работает также, как и спецификатор private. Но, различие заключается в том,

что методы и поля содержащиеся в данной секции, могут быть употреблены в качестве данных классами наследниками как-будто

они содержатся в секции public.<br><br>

<i> Пример: </i>

<div style="padding-left: 15px">

<code>

class CLASS\_BASE{<br>

&nbsp;&nbsp;protected:<br>

&nbsp;&nbsp;&nbsp;//начало секции protected<br>

&nbsp;&nbsp;&nbsp;int a;<br>

&nbsp;&nbsp;&nbsp;double b;<br>

&nbsp;&nbsp;&nbsp;//конец секции protected<br>

};<br><br>

class CLASS\_A : public CLASS\_BASE{<br>

&nbsp;&nbsp;public:<br>

&nbsp;&nbsp;&nbsp;int get\_a(){<br>

&nbsp;&nbsp;&nbsp;&nbsp;return CLASS\_BASE::a;<br>

&nbsp;&nbsp;}<br>

}<br><br>

void function(){<br>

&nbsp;&nbsp;CLASS\_BASE object;<br>

&nbsp;&nbsp;object.a = 90;//ERROR<br>

&nbsp;&nbsp;object.b = 120.312//ERROR<br>

&nbsp;&nbsp;CLASS\_A pobj;<br>

&nbsp;&nbsp;pobj.a = 90;<br>

&nbsp;&nbsp;pobj.b = 120.312<br>

&nbsp;&nbsp;pobj.get\_a()<br>

}<br><br><br>

</code>

</div>

Как видно из примера объект класса CLASS\_BASE не может обратиться к полям a и b, но объект класса наследника класса CLASS\_BASE,

может это сделать и изменять значения данных, значения которых изменять не может object.<br>

Думаю суть спецификаторов доступа понятна.

<br><br>

На этом данная статья заканчивается, и я надеюсь читателю стало понятна синтаксически что такое класс и объект.

<br><br>

</td>

</tr>

</tbody>

</table>

</div>

<div>

<table class="class\_table\_down" cellspacing="0">

<tbody>

<td class="class\_cell\_down">

<br>Контактная информация: <a href="https://vk.com/id560798514"> <img src="vk.jpg" height="35px" width="40px" alt="Страница в ВКонтакте" style="vertical-align: middle"> </a>

<br><br>

</td>

<td class="class\_cell\_down">

<br>

<right>

<a href="class\_object\_technical.html"> <img src="up\_point.png" height="35px" width="40px"> </a>

</right>

</td>

</tbody>

</table>

</div>

</body>

</html>

Исходный текст шаблона сайта

с использованием блоков (object\_oriented\_programming.html)

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link rel="stylesheet" href="DataCSS.css" type="text/css">

</head>

<body>

<div class="very\_top\_block"> <br>Введение в объектно-ориентированное программирование </div>

<div style="margin: 0px; width: 100%"></div>

<table class="class\_top\_table">

<tbody>

<tr>

<td style="width: 25%;">

<a href="base\_site.html"> Главная страница </a>

</td>

<td style="width: 25%;">

<a href="glossary.html"> Глоссарий </a>

</td>

<td style="width: 25%;">

<a href="site\_map.html"> Карта сайта </a>

</td>

<td style="width: 25%;">

<a href="about\_author.html"> Об авторе </a>

</td>

</tr>

</tbody>

</table>

</div>

<div style="margin:0px; width: 100%">

<table class="class\_base\_table">

<tbody>

<tr>

<td class="class\_cell\_left">

<ul class="class\_ul">

<li> <a href="history.html"> История объектно-ориентированного программирования </a> </li><br>

<li> Объектно-ориентированное программирование </li><br>

<li> <a href="class\_object.html"> Классы и объекты </a></li><br>

<li> <a href="conception.html"> Концепции объектно-ориентированного программирования </a> </li><br>

<li> <a href="oop\_language.html"> Объектно-ориентированные языки </a></li><br>

</ul>

</td>

<td class="class\_cell\_right">

<div>

<table class="class\_table\_down">

<tbody>

<tr>

<td style="text-align: left;">

<a href="history.html"> <img src="left\_point.png" width="35" height="40" alt="Переход влево"> </a>

</td>

<td style="text-align: right;">

<a href="class\_object.html"> <img src="right\_point.png" width="35" height="40" alt="Переход вправо"> </a>

</td>

</tr>

</tbody>

</table>

</div>

<center>

<h1> Объектно-ориентированное программирование </h1>

</center>

<br>

<a href="glossary.html"> <em> Объектно-ориентированное программирование (ООП) </em> </a> является доминирующим стилем при создании больших программ. Основные этапы эволюции структурного подхода в программировании помогают лучше понять взаимосвязь структурного подхода, модульного программирования и ООП.

Удельная стоимость создания программ до последнего времени менялась мало. С ростом объема программы удельная стоимость ее создания могла нелинейно возрастать. Время создания сложных программ пропорционально квадрату или даже кубу объема программ.

Поэтому одним из основных факторов, определяющих развитие технологии программирования, является снижение стоимости проектирования и создания программных продуктов (ПП) или борьба со сложностью программирования.

<br><br>

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

<br>

<div style="padding-left: 10px; text-align: left">

- изменение архитектур вычислительных средств (ВС) в интересах повышения

производительности, надежности;<br>

- упрощение взаимодействия пользователей с ВС и интеллектуализация ВС.

</div>

<br><br>

Действие двух последних факторов 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).

<br><br>

Выделяют 5-ть источников сложности программирования:

<br>

<div style="padding-left: 10px; text-align: left">

- решаемая задача;<br>

- язык программирования;<br>

- среда выполнения программы;<br>

- технологический процесс коллективной разработки и создания ПП;<br>

- стремление к универсальности и эффективности алгоритмов и типов данных.<br>

</div> <br>

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

В программировании широко используется фундаментальный принцип управления сложными системами, который известен человеку с глубокой древности - devide et impera (разделяй и властвуй, лат.)

и применяется при разработке и проектировании любых сложных технических систем. Согласно первой части этого принципа,

при проектировании сложной программной системы проводится алгоритмическая декомпозиция решаемой задачи.<br>

Целью декомпозиции является представление разрабатываемой системы в виде взаимодействующих небольших подсистем (модулей или блоков), каждую из которых можно отладить независимо от других.

При разработке системы, разделенной на подсистемы, необходимо держать в уме информацию о гораздо меньшем числе деталей, чем в отсутствие такого разделения.

<br><br>

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 и функций, написанных в процедурно-ориентированном стиле. Данный стиль опирается на правило:

определить переменные и константы, которые понадобится хранить в памяти компьютера и описать алгоритм их обработки.

<br><br>

Теоретическое оформление структурный подход получил в начале 70-х годов в работах теоретиков и практиков программирования(А.П.Ершова, Э. Йодана, Н.Вирта).

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.

<br><br>

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 связь модульного программирования и ООП.

<br><br>

Важнейшими инструментами производителей ПП, в которых находят отражение практически все аспекты эволюции, являются языки программирования.

<br><br>

Язык программирования изначально ориентирован на компьютер и содержит набор типов данных, операторов, операций, функций, которые достаточно просто могут быть переведены

в 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.

<br><br>

Развитие языков в направлении повышения эффективности составления прикладных программ привело к разделению языков по следующим уровням:<br>

<div style="padding-left: 10px; text-align: left">

- низкий уровень (машинно-ориентированные языки - языки ассемблера),<br>

- высокий уровень (процедурно-ориентированные языки: FORTRAN, ALGOL,

PL/1, Pascal)<br>

- уровень решаемой задачи (проблемно-ориентированные языки - SQL).<br>

</div><br>

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 функциями программных модулей,

и, во-вторых, сократить объем рутинной работы программиста при кодировании алгоритма обработки.<br><br>

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

<br><br>

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.<br>

Человек мыслит образами или объектами, он знает их свойства и манипулирует ими, сообразуясь с определенными событиями.

Древним грекам принадлежала мысль о том, что мир можно рассматривать в виде объектов и событий.

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 технических знаний, фантазии).<br><br>

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 на математические

объекты, на определенную модель вычислителя. Поэтому они содержали такие конструкции как переменная, константа, функция,

формальные и фактические параметры. Программисты представляли свои программы в виде взаимодействующих функций и модулей.

Характер программирования был процедурно-ориентированным, поскольку первостепенное внимание уделялось последовательностям

действий с данными. Соответственно такие языки программирования, как FORTRAN, PL-1, С называют процедурно-ориентированными.

<br><br>

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. Так, модуль обработки строк содержал функции выполнения основных операций со строками: объединения,

сравнения, копирования, вычисления длины строки.

<br><br>

Развитием идеи модульного программирования является сопоставление объектам предметной области (моделируемым объектам) программных конструкций, н

зываемых объектами, объектными типами или классами (моделирующими объектами). Моделирующие объекты содержат данные и функции, которые

описывают свойства моделируемых объектов. Так, данные могут отражать признаковые или количественные свойства

(масса, длина, мощность, цена), а функции отражают поведенческие или операционные свойства (изменить массу, вычислить мощность,

установить цену). Таким образом, при объектном подходе интеграция данных и функций их обработки определяется структурой предметной области,

т.е набором моделируемых объектов, их взаимодействием в рамках решаемой задачи.

<br><br>

Моделируемый объект всегда представляется человеку чем-то единым, целостным, хотя может состоять из частей или других объектов. Целостное

представление объекта в виде взаимосвязанной совокупности его свойств или компонентов является базовым принципом объектного подхода.

<br><br>

Объектный подход начал развиваться в программировании с 70-х годов (Smalltalk, CLOS, Ada). Эти языки называются объектными.

Иерархическая классификация объектов инаследование свойств являются отправными идеями появившегося в 80-х годах объектно-ориентированного

подхода. Одной из причин сравнительно медленного становления объектно-ориентированного стиля программирования является его существенное

отличие от процедурно-ориентированного стиля.

<br><br>

<left> Ссылка на источник: </left><br>

<a href="https://studopedia.ru/1\_84610\_strukturniy-podhod-v-programmirovanii.html" style="font-weight: bolder; padding-left: 10px;"> Студопедия </a>

<br><br>

</td>

</tr>

</tbody>

</table>

</div>

<div>

<table class="class\_table\_down" cellspacing="0">

<tbody>

<td class="class\_cell\_down">

<br>Контактная информация: <a href="https://vk.com/id560798514"> <img src="vk.jpg" height="35px" width="40px" alt="Страница в ВКонтакте" style="vertical-align: middle"> </a>

<br><br>

</td>

<td class="class\_cell\_down">

<br>

<right>

<a href="object\_oriented\_programming.html"> <img src="up\_point.png" height="35px" width="40px"> </a>

</right>

</td>

</tbody>

</table>

</div>

</body>

</html>

Исходный код файла каскадных таблиц (DataCSS.css)

body{

background-color: white;

}

.very\_top\_block{

min-height: 80px;

max-height: 200px;

height: auto;

width: auto;

background-color: #4d524e;

background-attachment: fixed;

text-align: left;

color: #b9b0b9;

font-family: Arial, 'Times New Roman', sans-serif;

font-size: 220%;

font-weight: bolder;

padding-left: 25px;

padding-top: 5px;

padding-bottom: 7px;

}

.class\_top\_table{

margin: 0;

min-height: 20px;

max-height: 40px;

height: auto;

width: 100%;

background-color: #d0e282;

text-align: center;

color: #5f5353;

font-family: Arial, 'Times New Roman', sans-serif;

font-size: 100%;

font-weight: bolder;

border-width: 0px;

border-style: none;

}

.class\_base\_table{

margin: 0;

min-height: 200px;

height: auto;

width: 100%;

border-width: 0px;

border-style: none;

}

.class\_cell\_left{

background-color: #ddd3d3;

color: #5f5353;

width: 20%;

padding-left: 10px;

padding-top: 5px;

text-align: left;

vertical-align: top;

font-family: Arial, 'Times New Roman', sans-serif;

font-weight: bolder;

}

.class\_cell\_right{

background-color: #f3efef;

color: #000000;

padding-left: 5px;

padding-top: 5px;

padding-bottom: 10px;

text-align: left;

font-family: Arial, 'Times New Roman', sans-serif;

width: auto;

min-width: 100px;

overflow: auto;

}

.class\_table\_down{

margin: 0;

min-height: 40px;

height: auto;

width: 100%;

border-width: 0px;

border-style: none;

}

.class\_table\_pointer{

margin: 0;

min-height: 40px;

height: auto;

width: 100%;

border-width: 0px;

border-style: none;

}

.class\_cell\_down{

background-color: #cabbbb;

color: #6d4a4a;

font-weight: bolder;

width: 100%;

text-align: center;

vertical-align: middle;

font-family: Arial, 'Times New Roman', sans-serif;

}

.class\_ul{

list-style-type: square;

list-style-position: outside;

}

.class\_ul\_ul{

list-style-type: circle;

list-style-position: outside;

}

em{

font-weight: bolder;

}

/\* Ссылки \*/

A:link {

text-decoration: none;

color: #2e2c2c;

}

A:visited {

text-decoration: none;

color: #2e2c2c;

}

A:active {

text-decoration: none;

color: #2e2c2c;

}

A:hover {

text-decoration: underline;

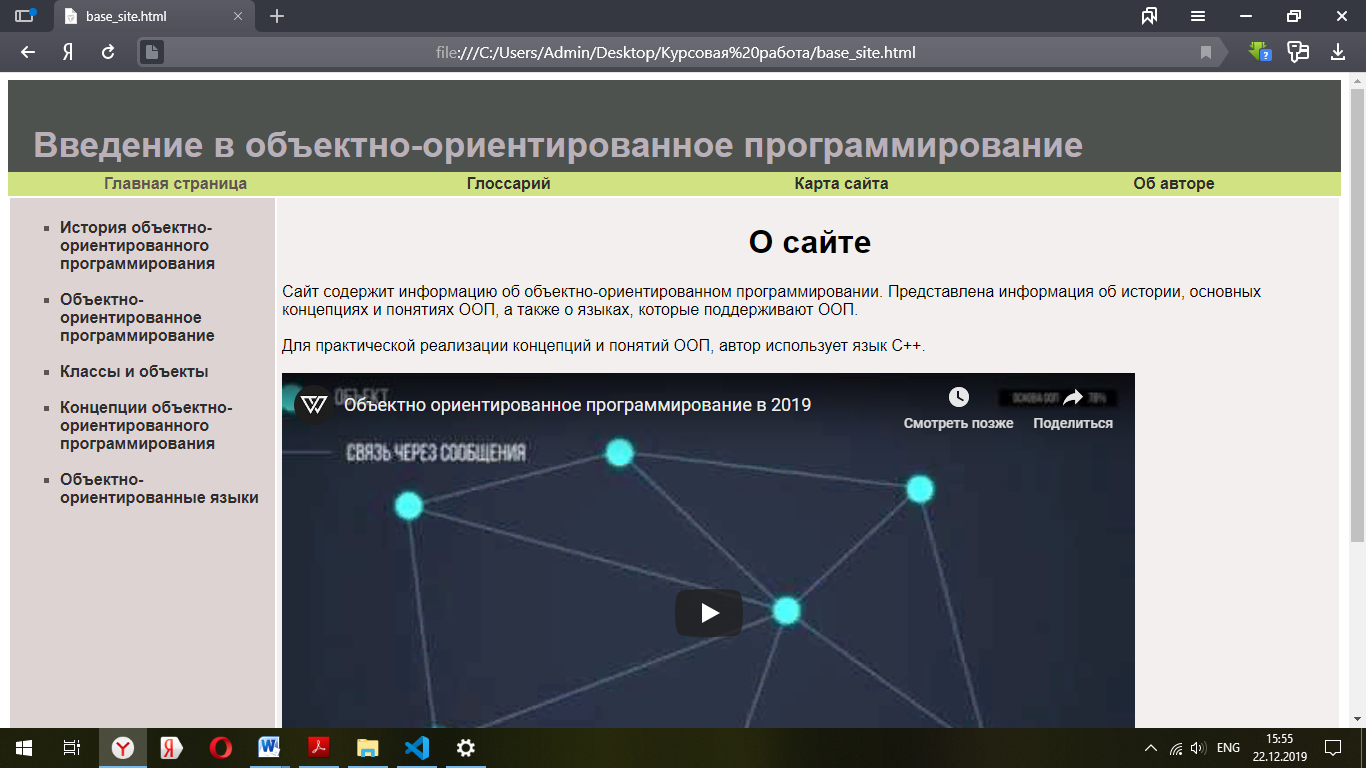
color: #67bfda;

}

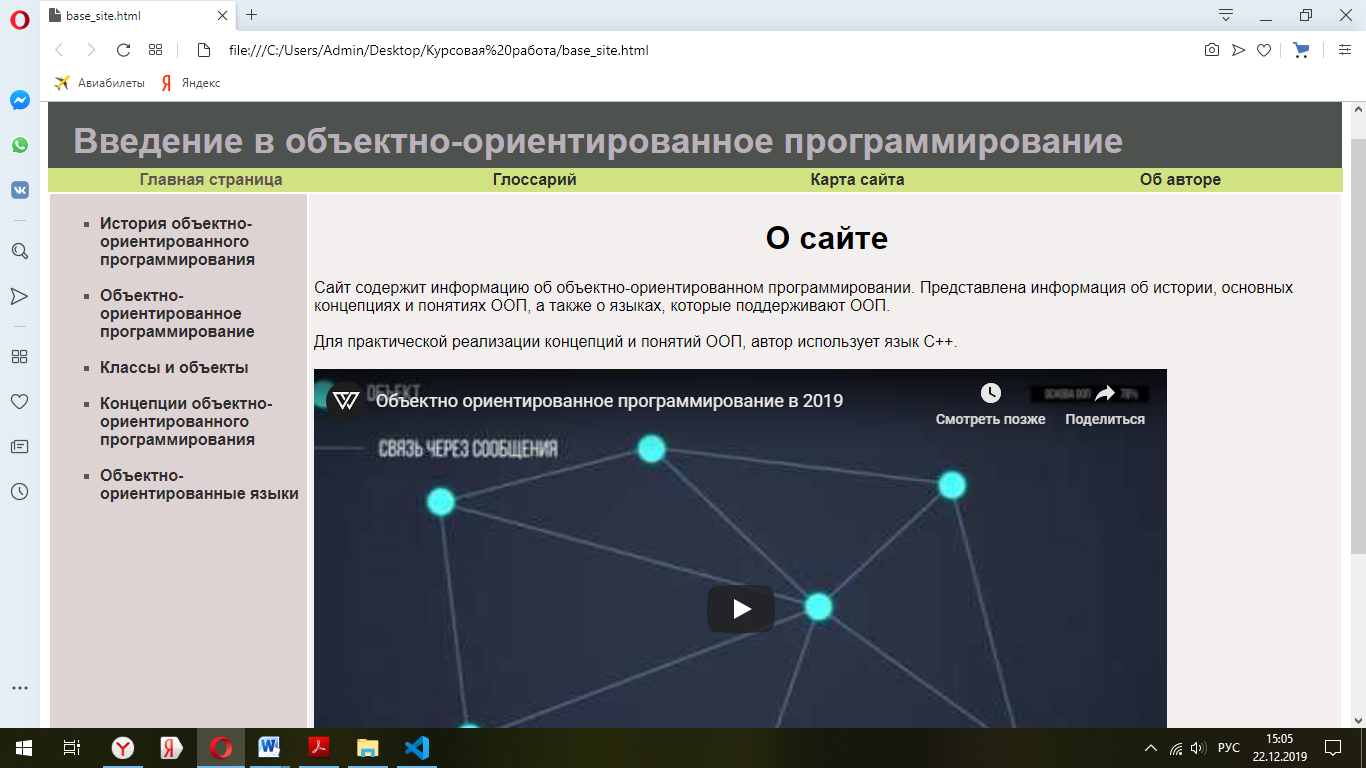
2.4 Тестирование сайта:

Тесты показали, что сайт оптимизирован для работы в различных браузерах:

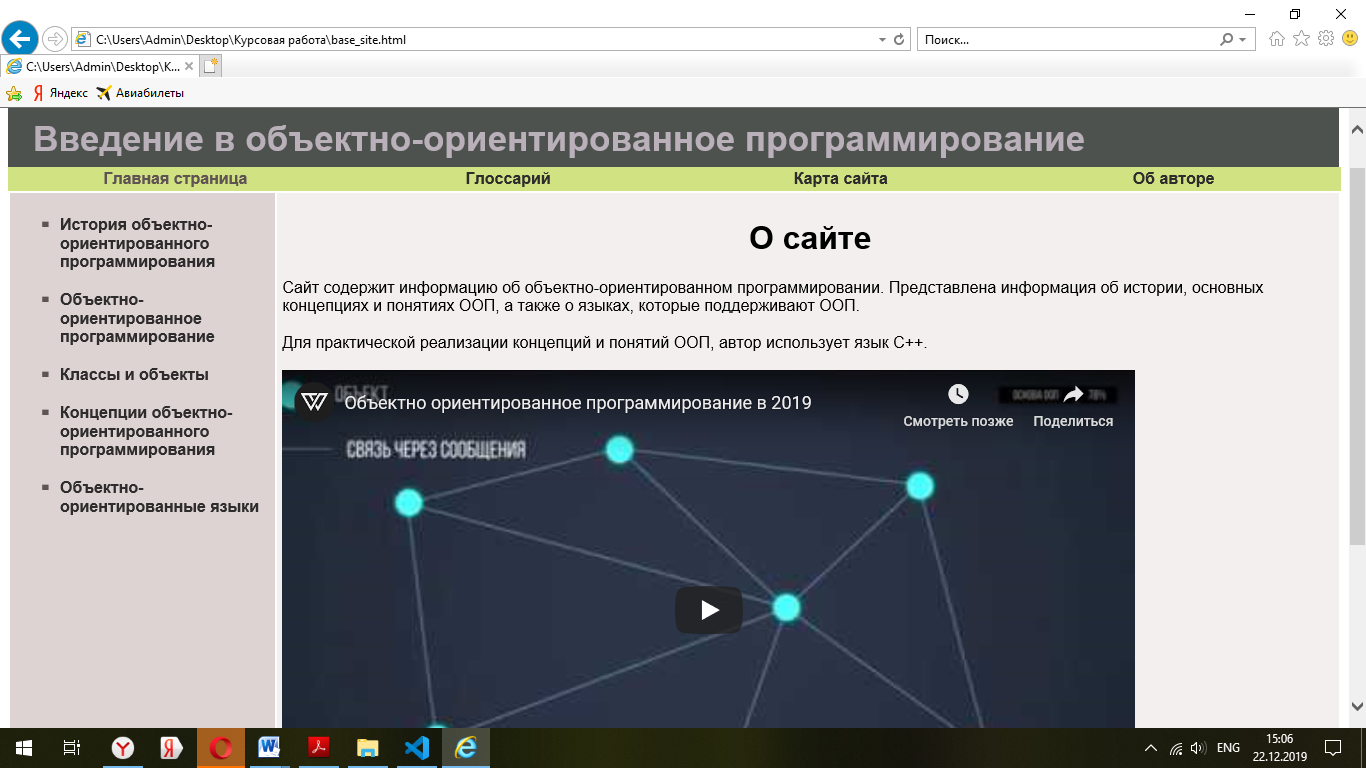
1. Яндекс с разрешением (1366x768)

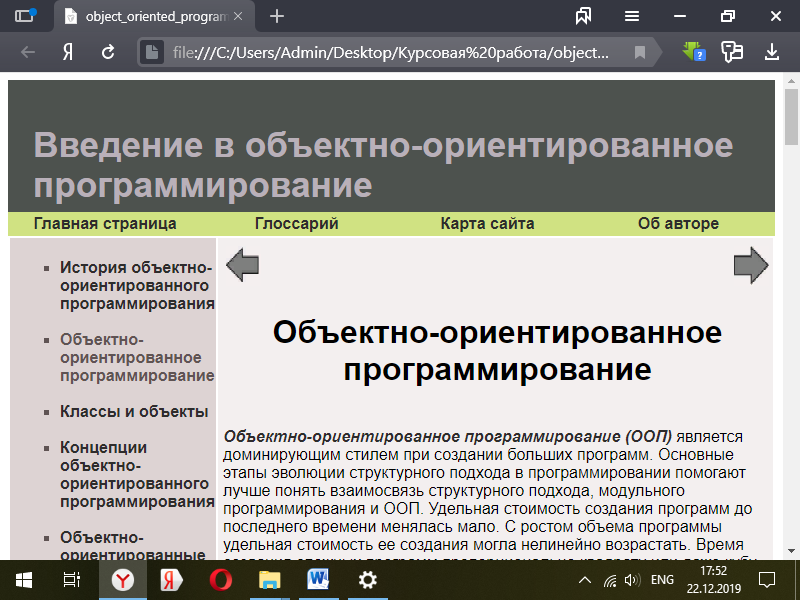


2. Opera с разрешением (1366x768)

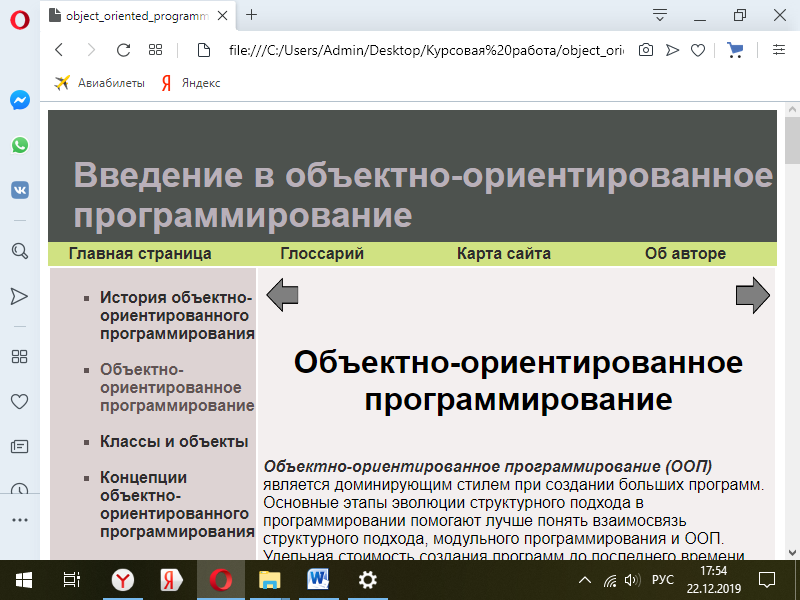


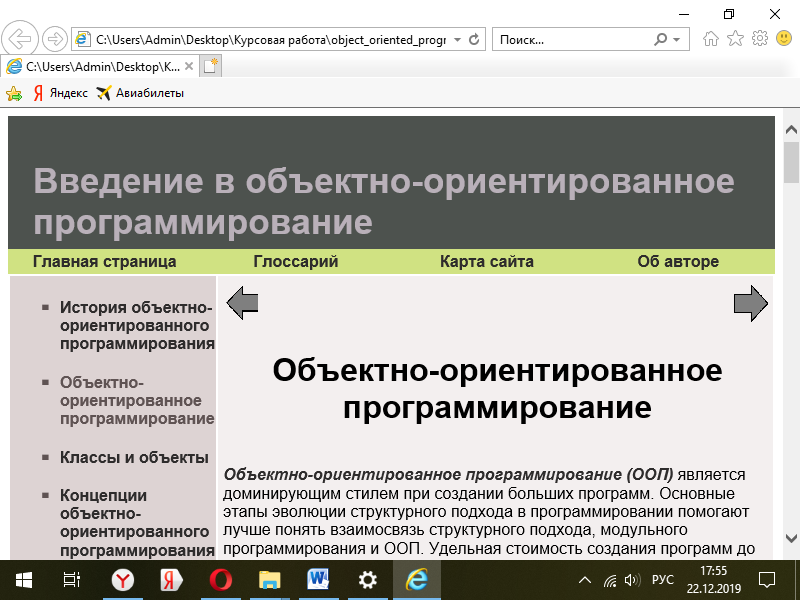
3. Internet Explorer с разрешением (1366x866)



4. Яндекс с разрешением (800 x 600)

5. Opera с разрешением (800x600)



6. Internet Explorer с разрешением (800x600)

2.5 Порядок разработки сайта

1. Подбор и обработка информации.

2. Построение логической структуры сайта.

3. Разработка структуры расположения всех файлов сайта.

3.1. Разработка иерархической структуры папок.

3.2. Создание и именование папок и файлов.

4. Разработка дизайна и проектирование сайта.

5. Верстка html-страниц.

5.1. Подбор редакторов кода и изображений

5.2. Набор кода.

5.3. Заполнение сайта информацией

6. Тестирование. Проверка функциональности и наполнения сайта.

6.1. Тестирование сайта в различных браузерах.

6.2. Формулирование вывода из тестирования.

7. Составление подробной документации по сайту и подготовка отчета.

2.6 Использованное ПО

Во время выполнения курсовой работы были использованы следующее ПО:

1.Visual Studio Code. Отличная программа для редактирования HTML-кода.

2. Paint. Использовал для редактирования изображений.

**Заключение**

При разработки сайта мной были получены базовые знания языка гипертекстовой разметки HTML и опыт в создании отчёта к курсовой работе. Данная курсовая работа послужила введением в web-разработку, поскольку дала нам некоторые теоретические и практические знания и навыки. Из проделанной работы можно сделать следующие выводы: перед написанием кода нужно установить структуру и дизайн web-сайта, затем изучить необходимую теорию и уже после – начать разрабатывать web-сайт.

Список используемых источников

1. Курсовая работа «Проектирование и реализация HTML документа» Руководство и методические указания для студентов направлений: 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии». Составители Бахвалова З.А., Серышева И.А. – Иркутск, Изд-во ИРНИТУ, 2017, 40с.
2. <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>
3. <https://habr.com/ru/post/444602/>
4. <https://studopedia.ru/1_84610_strukturniy-podhod-v-programmirovanii.html>
5. <https://habr.com/ru/post/445948/>
6. <https://www.youtube.com/embed/zqxVI_kEdq8>

Приложение А. Содержание сайта

История объектно-ориентированного программирования.

ООП возникло в результате развития идеологии процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны. Для дальнейшего развития объектно-ориентированного программирования часто большое значение имеют понятия события (так называемое событийно-ориентированное программирование) и компонента (компонентное программирование, КОП).

Взаимодействие объектов происходит посредством сообщений. Результатом дальнейшего развития ООП, по-видимому, будет агентно-ориентированое программирование, где агенты — независимые части кода на уровне выполнения. Взаимодействие агентов происходит посредством изменения среды, в которой они находятся.

Языковые конструкции, конструктивно не относящиеся непосредственно к объектам, но сопутствующие им для их безопасной (исключительные ситуации, проверки) и эффективной работы, инкапсулируются от них в аспекты (в аспектно-ориентированном программировании). Субъектно-ориентированное программирование расширяет понятие объекта посредством обеспечения более унифицированного и независимого взаимодействия объектов. Может являться переходной стадией между ООП и агентным программированием в части самостоятельного их взаимодействия.

Первым языком программирования, в котором были предложены основные понятия, впоследствии сложившиеся в парадигму, была Симула, но термин «объектная ориентированность» не использовался в контексте использования этого языка. В момент его появления в 1967 году в нём были предложены революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Фактически, Симула была «Алголом с классами», упрощающим выражение в процедурном программировании многих сложных концепций. Понятие класса в Симуле может быть полностью определено через композицию конструкций Алгола (то есть класс в Симуле — это нечто сложное, описываемое посредством примитивов).

Взгляд на программирование «под новым углом» (отличным от процедурного) предложили Алан Кэй и Дэн Ингаллс в языке Smalltalk. Здесь понятие класса стало основообразующей идеей для всех остальных конструкций языка (то есть класс в Смолтоке является примитивом, посредством которого описаны более сложные конструкции). Именно он стал первым широко распространённым объектно-ориентированным языком программирования.

В настоящее время количество прикладных языков программирования (список языков), реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. Наиболее распространённые в промышленности языки (С++, Delphi, C#, Java и др.) воплощают объектную модель Симулы. Примерами языков, опирающихся на модель Смолтока, являются Objective-C, Python, Ruby.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) является доминирующим стилем при создании больших программ. Основные этапы эволюции структурного подхода в программировании помогают лучше понять взаимосвязь структурного подхода, модульного программирования и ООП. Удельная стоимость создания программ до последнего времени менялась мало. С ростом объема программы удельная стоимость ее создания могла нелинейно возрастать. Время создания сложных программ пропорционально квадрату или даже кубу объема программ. Поэтому одним из основных факторов, определяющих развитие технологии программирования, является снижение стоимости проектирования и создания программных продуктов (ПП) или борьба со сложностью программирования.

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

- изменение архитектур вычислительных средств (ВС) в интересах повышения производительности, надежности;

- упрощение взаимодействия пользователей с ВС и интеллектуализация ВС.

Действие двух последних факторов 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).

Выделяют 5-ть источников сложности программирования:

- решаемая задача;

- язык программирования;

- среда выполнения программы;

- технологический процесс коллективной разработки и создания ПП;

- 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 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 независимо от других. При разработке системы, разделенной на 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 и функций, написанных в процедурно-ориентированном стиле. Данный стиль опирается на правило: определить переменные и константы, которые понадобится хранить в памяти компьютера и описать алгоритм их обработки.

Теоретическое оформление структурный подход получил в начале 70-х годов в работах теоретиков и практиков программирования(А.П.Ершова, Э. Йодана, Н.Вирта). 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 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, которые достаточно просто могут быть переведены в 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:

- низкий уровень (машинно-ориентированные языки - языки ассемблера),

- высокий уровень (процедурно-ориентированные языки: FORTRAN, ALGOL, PL/1, Pascal)

- уровень решаемой задачи (проблемно-ориентированные языки - 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 функциями программных модулей, и, во-вторых, сократить объем рутинной работы программиста при кодировании алгоритма обработки.

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

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. Древним грекам принадлежала мысль о том, что мир можно рассматривать в виде объектов и событий. 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 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 на математические объекты, на определенную модель вычислителя. Поэтому они содержали такие конструкции как переменная, константа, функция, формальные и фактические параметры. Программисты представляли свои программы в виде взаимодействующих функций и модулей. Характер программирования был процедурно-ориентированным, поскольку первостепенное внимание уделялось последовательностям действий с данными. Соответственно такие языки программирования, как FORTRAN, PL-1, С 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. Так, модуль обработки строк содержал функции выполнения основных операций со строками: объединения, сравнения, копирования, вычисления длины строки.

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

Моделируемый объект всегда представляется человеку чем-то единым, целостным, хотя может состоять из частей или других объектов. Целостное представление объекта в виде взаимосвязанной совокупности его свойств или компонентов является базовым принципом объектного подхода.

Объектный подход начал развиваться в программировании с 70-х годов (Smalltalk, CLOS, Ada). Эти языки называются объектными. Иерархическая классификация объектов инаследование свойств являются отправными идеями появившегося в 80-х годах объектно-ориентированного подхода. Одной из причин сравнительно медленного становления объектно-ориентированного стиля программирования является его существенное отличие от процедурно-ориентированного стиля.

**Классы и объекты.**

Понятие класса и объекта

Класс и объект являются одними из самых важных концепций объектно-ориентированного программирования, и прежде чем приступить к изучению парадигм и других концепций ООП, следует понять что такое класс и объект. В википедии понятию класс и объект даны следующие определения:

Класс — универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей).

Объект - сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Сейчас, я сделаю попытку раскрыть эти термины наиболее понятным языком. Начнём с вопроса первенства создания класса или объекта, а именно: что первое? Класс или объект? Однозначный ответ на этот вопрос: класс. Класс может существовать и без объекта, однако объект не может существовать без класса, отсюда сильная зависимость объекта от класса. Класс позволяет однозначно определить объект, задать его атрибуты, определить его действия, то есть выделяет один объект из множества других возможных объектов, это в свою очередь придаёт объекту некоторую уникальность. Здесь и далее “атрибуты” следует понимать как “поля”, а “поведения” как “методы”, эти наименования идентичны в рамках ООП. Но понятие “поведение” используется крайне редко. Если давать понятие объекту в данном контексте, но ориентируясь на физический мир, то объект представляет собой продукт взаимодействия среды, которая создаёт что-либо, и класса (описание того, что и как среда будет создавать). Объект в таком понимании легко находит аналоги в физическом мире.

Например, таким объектом можно считать автомобиль. Автомобиль берёт своё начало на заводе, который создаёт автомобиль путём проектирования и использования определённых знаний в автомобилестроении. Ключевое понятие здесь – проектирование автомобиля. После проектирования идёт процесс создания автомобиля, а затем, автомобиль представляет собой целостный объект, который был получен при создании согласно проектированию. Среда же, которая путём взаимодействия с классом (описанием) будущего объекта, является самим заводом по созданию автомобилей. После создания автомобиля, завод, скорее всего, будет заинтересован в том, чтобы автомобиль был в целости и сохранности, поэтому поместит его на склад, где и будет храниться этот автомобиль какое-то время. Т.е. объект будет занимать какое-то место на складе. А теперь, останавливаясь на заводе, перейдём в абстрактный мир вычислительной машины и обобщим наш пример с автомобильным заводом. Пройдёмся ещё раз по времени жизни проекта автомобиля до его хранения как целостного объекта на заводе.

Код C++ :

Порядок:

1) Проектирование автомобиля (создание его описания)

2) Создание автомобиля пользуясь результатами проектирования (по описанию)

3) Хранение автомобиля на складе (автомобиль занимает какое-то место)

Теперь, пожалуй, для придания практичности этому разделу, напишем код, который будет создавать объект автомобиля. По этому коду будем разбираться с обобщением нашего примера. Язык программирования, который будет использоваться с целью практического созерцания объектно-ориентированного программирования, называется C++.

#include <iostream>

#include <string>

//Класс лишён углублённого описание технических характеристик и тонкостей

//которые необходимы при создании автомобиля.

class Autocar { //POINT A

private:

char count\_wheel; //количество колёс

double price; //цена

std::string title; //название автомобиля

std::string title\_plant; //название завода

std::string country; //страна

public:

void Set\_Attribute(char cw, double p, std::string t1, std::string t2, std::string c) {

count\_wheel = cw;

price = p;

title = t1;

title\_plant = t2;

country = c;

return;

}

void ShowInConsole\_Attribute() {

std::cout << "Name car : " << title.data() << '\n'\

<< "Name plant : " << title\_plant.data() << '\n'\

<< "Country : " << country.data() << '\n'\

<< "Count wheel : " << (short)(count\_wheel) << '\n'\

<< "Price : " << price << "\n\n";

return;

}

}; //POINT B

int main(){

Autocar autocar1; // Создание объекта

autocar1.Set\_Attribute(4, 2000, "Audi", "Plant by create car", "empty"); //Придание объекту атрибутов

autocar1.ShowInConsole\_Attribute(); //Вывод информации об объекте на консоль

system("PAUSE");

return 0;

}

Разберём код. В самом начале идёт подключение необходимых библиотек: iostream для использования вывода на консоль необходимой информации и string для хранения строк. Далее между комментариями POINT A и POINT B содержится код определения класса Autocar. Любое определение класса начинается с ключевого слова class, за которым идёт имя класса а затем блок кода содержащий описание класса, с завершающей точкой с запятой (;). Блок определения класса содержит атрибуты, представленные ,в данном случае, в виде переменных count\_wheel, title, title\_plant, country. Атрибуты и поля класса это одно и тоже, как было упомянуто выше. Вообще как интерпретировать атрибуты класса? Атрибутами класса могут быть константы, статические переменные, константы, указатели, указатели на функции, переменные любых типов данных и прочие конструкции позволяющие обрабатывать данные. Т.е. атрибуты в первую очередь данные, а потом уже являются характеристикой объекта в абстрактном понимании, хотя две этих вещи разделять не стоит, но очерёдность здесь уместна лишь из-за выявления очень прозрачной границы между атрибутами как данными и характеристикой объекта, но граница очень прозрачна, а сделано это для понимания.

Атрибутами можно манипулировать, с помощью методов класса. Методы класса могут быть вызваны объектом класса, который является инициатором начала манипуляций с атрибутами класса. Т.е. объект сам изменяет свои атрибуты, но только при определённых условиях или действиях, которые инициирует сам, и, конечно, стоит учесть очевидный факт, что если атрибут имеет спецификатор const (константная переменная например), то никакой речи идти не может о том, чтобы объект каким-либо образом повлиял на данный атрибут. Такие атрибуты только используются методами класса. Вообщем, методы класса – это функции, которые, при вызове объектом класса, оперируют атрибутами класса (его полями). В примере методы класса определены функциями Set\_Attribute() и ShowInConsole\_Attribute(). Один метод класса задаёт данные атрибутам путём присваивания, а другой выводит на консоль данных полей, предварительно определив факт принадлежности конкретного числа конкретному описанию (какая страна, какова цена, название машины и завода и количество колёс). Класс определяет атрибуты, а объект использует их. Теперь вернёмся к коду и нашему примеру. Класс Autocar определён и теперь мы можем создать на основе этого класса объект. Здесь стоит вспомнить пример с автозаводом. Ведь действия все происходят на автозаводе, а именно: создание автомобиля, транспортировка его на склад и последующее хранение. Представьте что main() это завод. Действия завода ограничены самим блоком кода функции main().

В C++ работа программы всегда начинается с функции main(). По крайней мере, это истинно для обобщённого созерцания работы программы и программного кода. Итак, main() это завод. Первый оператор в функции main(), который будет встречен при анализе кода сразу после фигурной скобки, идентифицирующей в пространстве текста программы начало блока кода функции main(), да и вообще, любой функции, будет являться объявлением локальной статической переменной типа Autocar, или просто – созданием объекта. Так вот, если соединить наш пример с пониманием области видимости любой функции, то получится достаточно интересная вещь. После создания объекта в функции main(), этот объект будет являться статической локальной переменной типа Autocar, что не нарушает терминологии взятой из википедии, а из этого следует, что эта переменная будет занимать определённый объём памяти компьютера. А теперь свяжите создание и хранение переменной autocar1 с созданием и последующим хранением автомобиля на складе. Чувствуете сильную аналогию? Только компьютер реализует это абстрактно, придерживаясь определённых технических установок, а реальный автозавод реализует это в физическом мире. Если и дальше продолжить анализ кода, то мы увидим что объект вызывает методы Set\_Attribute() и ShowInConsole\_Attribute(), первый из них задаёт новые данные атрибутам объекта класса Autocar, а второй выводит данные об атрибутах на консоль. Затем происходит небольшая задержка, после которой будет встречен оператор return.

Оператор return идентифицирует точку выхода из функции, а , следовательно, для main() это означает ещё и завершение программы. Но перед полным завершением программы освобождается локальная память, и статическая локальная переменная autocar1 (объект класса Autocar) уничтожается. Данный факт взят из понятия об области видимости переменных, которые определены функции. Уничтожение переменной autocar1, свяжите с вывозом из склада автомобиля на распродажу или утилизацию, но в любом случае, autocar1 на складе уже не будет.

На этом данная тема заканчивается, и я надеюсь читателю было понятно что такое класс, что такое объект и как они между собой взаимосвязаны и что от чего зависит. Техническая информация на эту тему, будет также представлена в соседнем разделе под названием “Класс и объект. Конструкции языка” в котором подробнейшим образом разберётся техническая составляющая класса и объекта. Ниже будут ссылки на сайты, в которых можно узнать по подробнее о области определения функции, переменных и всего остального, что было упомянуто выше и вызвало у вас возможные затруднения из-за незнания или успешно забытого материала.

1. <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C_%D0%B2%D0%B8%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8>
2. <https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>

Классы и объекты. Конструкции языка

В предыдущей статье была дана теоретическая информация относительно объектов и классов. В этой статье будет дано описание объектов и классов с точки зрения конструкций языка ну и практики как таковой.

**Класс**

В синтаксис определения класса входит ключевое слово class, имя класса и блок кода, в котором определяются поля и методы.

Синтаксис определение класса:

class name\_class{

//блок кода

}

Объявление объекта класса в блоке кода любой функции будет выглядеть следующим образом:

name\_class object;

Как видно из синтаксиса объявления объекта класса name\_class object, синтаксис очень похож на объявление переменных. Здесь стоит обратить внимание на определение класса из википедии. Класс очень схож с типом данных, и отсюда явно видна аналогия в синтаксисе между объявлением простой переменной и объявлением объекта(переменной сложного типа).

**Спецификаторы доступа**

В блоке кода определения класса методы и поля разделяются на три секции: public, private, protected

Зачем вообще такое разделение? Каждый объект класса имеет возможность обращаться к определённым полям и методам. Спецификаторы доступа управляют доступом объекта к полям и методам, хранящимся в определённой секции.

В секцию входит спецификатор, множество состоящее из операторов объявления полей и объявления/определения методов, и сама секция идёт пока не встретит завершения блока кода определения класса или другой спецификатор.

Пример:

class CLASS{

public:

//начало секции public

int a;

int b;

//конец секции public private:

//начало секции private

double a;

double c;

//конец секции private

}

В примере сделана попытка показать деление данных блока кода класса на секции. Сейчас мы рассмотрим описание всех спецификаторов доступа. То есть, если спецификаторы доступа ограничивают объекту доступ к данным находящимся в определённой секции, то каким образом они это делают? Сейчас мы и выясним.

**public**

Всё что находится в секции public является доступным объекту класса через операцию точки(.) или косвенного обращения(->). Т.е. объект класса имеет доступ к любым полям и методам класса, объектом которого он является, в котором эти поля и методы определены/объявлены.

Пример:

class CLASS{

public:

//начало секции public

int a;

double b;

//конец секции private

};

void function(){

CLASS object;

object.a = 90;

object.b = 120.312

}

Как видно из примера, объект класса CLASS object, без проблем имеет доступ к своим полям a и b, и изменяет их значения. Объект класса CLASS object обращается к полям и методам через операцию точки(.), или, в случае если объект - указатель, с помощью операции косвенного обращения(->).

**private**

Данный спецификатор ограничивает объекту класса доступ к полям и методам полностью. Т.е. объект класса не может обращаться к полями и методам класса, которые находятся в секции private. Полное ограничение.

Пример:

class CLASS{

private:

//начало секции private

int a;

double b;

//конец секции private

};

void function(){

CLASS object;

object.a = 90;//ERROR

object.b = 120.312//ERROR

}

В примере видно, что при обращении к полям класса CLASS, объект object обращается к полям которые находятся в секции private, и результат таких действий - ошибка.

**protected**

Данный спецификатор для обычного объекта работает также, как и спецификатор private. Но, различие заключается в том, что методы и поля содержащиеся в данной секции, могут быть употреблены в качестве данных классами наследниками как-будто они содержатся в секции public.

Пример:

class CLASS\_BASE{

protected:

//начало секции protected

int a;

double b;

//конец секции protected

};

class CLASS\_A : public CLASS\_BASE{

public:

int get\_a(){

return CLASS\_BASE::a;

}

}

void function(){

CLASS\_BASE object;

object.a = 90;//ERROR

object.b = 120.312//ERROR

CLASS\_A pobj;

pobj.a = 90;

pobj.b = 120.312

pobj.get\_a()

}

Как видно из примера объект класса CLASS\_BASE не может обратиться к полям a и b, но объект класса наследника класса CLASS\_BASE, может это сделать и изменять значения данных, значения которых изменять не может object.

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 показали, 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 называют 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.

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 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.

**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 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.

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

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 отличаются именами.

**Инкапсуляция свойств объектов.**

Инкапсуляция ( «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 соответствующие имена полей и методов при помощи операций выбора «.» и «->». Это позволяет в максимальной степени изолировать содержание объекта от внешнего окружения, т. е. ограничить и наглядно контролировать доступ к элементам объекта. В результате замена или модификация полей и методов, инкапсулированных в объект, как правило, не влечет за собой плохо контролируемых последствий 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.

Зачем нужна инкапсуляция ? Ответ прост, мы - люди. А человеку свойственно ошибаться. Никто не застрахован от ошибок. Применяя инкапсуляцию, мы, как бы, возводим крепость, которая защищает данные, принадлежащие объекту, от возможных ошибок, которые могут возникнуть при прямом доступе к этим данным. Кроме того, применение этого принципа очень часто помогает локализовать возможные ошибки в коде программы. А это на много упрощает процесс поиска и исправления этих ошибок.

Можно сказать, что инкапсуляция подразумевает под собой скрытие данных (data hiding), что позволяет защитить эти данные.

А теперь определение, которое точно определяет суть инкапсуляции:

Переменные состояния объекта скрыты от внешнего мира. Изменение состояния объекта(его переменных) возможно ТОЛЬКО с омощью его методов(операций).

Почему же это так важно? Этот принцип позволяет защитить переменные состояния объекта от неправильного их использования.

Это существенно ограничивает возможность введения объекта в недопустимое состояние и несанкционированное разрушение этого объекта.

Для иллюстрации приведенного выше постулата рассмотрим пример.

Представьте, что у Вас не заводится машина и Вы, увы, не механик и плохо разбираетесь в машинах. Вы открываете капот и начинаете выдергивать какие-то шланги, что-то окручивать и т.д. Хорошо, если Вы запомнили что, где и как выдергивали и откручивали. А если нет? Или у Вас стрелка уровня топлива стоит на нуле, а Вы считаете, что у Вас полно топлива и полезете со спичками внутрь бензобака проверять уровень топлива. Какие последствия Вас могут ожидать? В лучшем случае Вы и Ваша машина останутся живы, если Вам очень повезет. Аналогично и с нашими объектами, которые могут быть чрезвычайно сложными, а Вы пытаетесь что-то в них подправить, не представляя их внутреннюю организацию.

Для того, чтобы починить машину не причинив себе и самой машине вреда необходимо пригласить квалифицированных авто-слесарей, причем каждый из которых отлично разбирается только в определенной части Вашей машины. Если Вы скажете, что у Вас не горит лампочка подсветки в салоне, то замену лампочки проведет специалист по электрооборудованию автомобилей. Аналогично и в нашем объекте. Есть "мастера" - методы, которые "специализируются" в определенных областях, но свою область они знают. А самое главное, они знают как можно изменить состояние объекта так, чтобы не повредить его. Описанный постулат отражает простую житейскую мудрость: не знаешь, не представляешь как что-то сделать - попроси это сделать того, кто знает как это правильно надо сделать. К сожалению, все мы на каждом шагу пренебрегаем этим правилом. В ООП мы это правило определяем как закон: "Объект не приемлет дилетантов. Только специалисты могут как-либо изменять состояние объекта." Вы можете сказать, что этот принцип далеко не новость в программировании.

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 окупятся при отладке и сопровождении продукта. Текст подобной программы становится понятным без особых комментариев. Дополнительным средством 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: частные, 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.

Методы в классе могут быть объявлены как дружественные (friend) или виртуальные (virtual). Иногда встречается объявление перегружаемых (overload) функций.

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. Это положение не являтся 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.

Приведем пример наследования из реальной жизни. Исследователи во многих областях естествознания тратят львиную долю времени на классификацию объектов в соответствии с определенными особенностями. Из школьного курса биологии и зоологии мы помним, что для животных, растений существуют специальные принципы классификации и разбиения на классы, подклассы, виды и подвиды (или что-то в этом роде) и т.д. В результате образуется своего рода иерархия или дерево с одной общей категорией в корне и подкатегориями, разветвляющимися на подкатегории.

Пытаясь провести классификацию некоторых новых животных или объектов, мы задаем следующие вопросы: В чем сходство этого объекта с другими объектами общего класса? В чем различия? Каждый класс имеет набор поведений и характеристик, которые его определяют. Мы начинаем с верхушки фамильного дерева образца и будем спускаться по ветвям, задавая эти вопросы на протяжении всего пути. Более высокие уровни являются более общими, а вопросы более простыми: например, есть крылья или нет крыльев? Каждый уровень является более специфическим, чем предыдущий уровень и менее общим. Когда характеристика определена, все категории ниже этого определения включают эту характеристику. Поэтому, когда мы говорим про того или иного конкретного представителя класса(отряда, вида), то нам не надо говорить про его общие особенности, характерные для этого класса, а говорим только про его специфические особенности в рамках этого класса.

Смысл и универсальность наследования заключается в том, что не надо каждый раз заново (с нуля) описывать новый объект, а можно указать родителя (базовый класс) и описать отличительные особенности нового класса. В результате, новый объект будет обладать всеми свойствами родительского класса плюс своими собственными отличительными особенностями.

Наследование в ООП позволяет адекватно отражать родственные отношения объектов предметной области. Если класс В обладает всеми свойствами класса А и еще имеет дополнительные свойства, то класс А называется базовым (родительским), а класс В называется наследником класса А. 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.

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 классыили 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.

При указании базового (родительского) класса в описании класса в С++ требуется указать ключевое слово public. Указание этого ключевого слова позволит получить свободный доступ ко всем методам класса, как если бы они были описаны в самом производном классе. В противном же случае, мы не сможем получить доступ к методам родительского класса.

Пример описания наследования классов на C++:

class A{

...

}

class B : public A{

...

}

Дополнительная информация о концепциях объектно-ориентированного программирования

Инкапсуляция

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, С++, Java или Ruby) отождествляют инкапсуляцию с сокрытием, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

По умолчанию, в классе (class) данные и методы приватные (private); они могут быть прочитаны и изменены только классом к которому принадлежат. Уровень доступа может быть изменен при помощи соответствующих ключевых слов которые предоставляет Си++. В Си++ доступно несколько спецификаторов, и они изменяют доступ к данным следующим образом:

* публичные (public) данные — доступны всем;
* защищенные (protected) — доступны только классу и дочерним классам;
* приватные (private) —доступны только классу которому они принадлежат.

Пример инкапсуляции

В классе Contact, публичные переменные и методы доступны из основной программы (main). Приватные переменные и методы могут прочитаны, вызваны или изменены только самим классом.

#include

using namespace std;

class Contact

{

private:

int mobile\_number; // private variable

int home\_number; // private variable

public:

Contact() // constructor

{

mobile\_number = 12345678;

home\_number = 87654321;

}

void print\_numbers()

{

cout << "Mobile number: " << mobile\_number;

cout << ", home number: " << home\_number << endl;

}

};

int main()

{

Contact Tony;

Tony.print\_numbers();

// cout << Tony.mobile\_number << endl;

// will cause compile time error

return 0;

}

Попытка напечатать или изменить приватную переменную mobile\_number из основной программы (main) вызовет ошибку при компиляции потому как доступ к приватным данным в классе ограничен.

В контексте инкапсуляции, все данные в Си могут быть рассмотрены как публичные по умолчанию. Уровень доступа к переменным в структурах (struct) может быть изменен на приватный если изолировать их определение от основной программы. Нужный эффект может быть достигнут при использовании отдельных заголовочных (header, .h) и исходных (source, .c) файлов.

Доступ к переменным в структуре ограничен. Тем не менее, спрятаны только переменные, не память в которой хранятся данные. Указатели можно рассматривать как ссылку на адрес памяти, и если эта память доступна программе — данные сохраненные в этой памяти можно прочитать и изменить. Если указатель назначен на память в которой структура хранит свои данные — их можно прочитать. Используя то же определение структуры (те же “.c” и “.h” файлы) и модифицированный “main.c” файл, ограничение доступа было преодолено.

Функции, будучи внешними (extern) по умолчанию, видимы во всей так называемой единице трансляции (translation unit). Другими словами, если несколько файлов скомпилированы вместе в один объектный файл, любой из этих файлов сможет получить доступ к любой функции из любого другого файла. Использование ключевого слова “статический” (static) при создании функции ограничит ее видимость до файла в котором она была определена. Следовательно, для обеспечения приватности функции необходимо выполнить несколько шагов:

функция должна быть объявлена статической (static) либо в исходном файле (.c), либо в соответствующем заголовочном файле (.h);

определение функции должно находиться в отдельном исходном файле.

Заключение Инкапсуляция существует за пределами ООП языков. Современные ООП языки делают использование инкапсуляции удобным и естественным. Существует множество способов обойти инкапсуляцию и избежание сомнительных практик поможет ее сохранить как в Си, так и в Си++.

Наследование

**Что такое наследование?**

Наследование является одним из основополагающих принципов ООП. В соответствии с ним, класс может использовать переменные и методы другого класса как свои собственные.

Класс, который наследует данные, называется подклассом (subclass), производным классом (derived class) или дочерним классом (child). Класс, от которого наследуются данные или методы, называется суперклассом (super class), базовым классом (base class) или родительским классом (parent). Термины “родительский” и “дочерний” чрезвычайно полезны для понимания наследования. Как ребенок получает характеристики своих родителей, производный класс получает методы и переменные базового класса.

Наследование полезно, поскольку оно позволяет структурировать и повторно использовать код, что, в свою очередь, может значительно ускорить процесс разработки. Несмотря на это, наследование следует использовать с осторожностью, поскольку большинство изменений в суперклассе затронут все подклассы, что может привести к непредвиденным последствиям.

В этом примере, метод turn\_on() и переменная serial\_number не были объявлены или определены в подклассе Computer. Однако их можно использовать, поскольку они унаследованы от базового класса.

Важное примечание: приватные переменные и методы не могут быть унаследованы.

Типы наследования

В C ++ есть несколько типов наследования:

* публичный (public)- публичные (public) и защищенные (protected) данные наследуются без изменения уровня доступа к ним;
* защищенный (protected) — все унаследованные данные становятся защищенными;
* приватный (private) — все унаследованные данные становятся приватными.

Для базового класса Device, уровень доступа к данным не изменяется, но поскольку производный класс Computer наследует данные как приватные, данные становятся приватными для класса Computer.

**Конструкторы и деструкторы**

В C ++ конструкторы и деструкторы не наследуются. Однако они вызываются, когда дочерний класс инициализирует свой объект. Конструкторы вызываются один за другим иерархически, начиная с базового класса и заканчивая последним производным классом. Деструкторы вызываются в обратном порядке.

Полиморфизм

Полиморфизм подтипов (в ООП называемый просто «полиморфизмом») — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма — параметрический — в ООП называют обобщённым программированием.

Существует несколько разновидностей полиморфизма. Две принципиально различных из них были описаны Кристофером Стрэчи в 1967 году: это параметрический полиморфизм и ad-hoc-полиморфизм, причём первая является истинной формой, а вторая — мнимой; прочие формы являются их подвидами или сочетаниями. Параметрический полиморфизм подразумевает исполнение одного и того же кода для всех допустимых типов аргументов, тогда как ad-hoc-полиморфизм подразумевает исполнение потенциально разного кода для каждого типа или подтипа аргумента. Бьёрн Страуструп определил полиморфизм как «один интерфейс — много реализаций», но это определение покрывает лишь ad-hoc-полиморфизм.

Принципиальная возможность для одного и того же кода обрабатывать данные разных типов определяется свойствами системы типов языка. С этой точки зрения различают статическую неполиморфную типизацию (потомки Алгола и BCPL), динамическую типизацию (потомки Lisp, Smalltalk, APL) и статическую полиморфную типизацию (потомки ML). Использование ad-hoc-полиморфизма наиболее характерно для неполиморфной типизации. Параметрический полиморфизм и динамическая типизация намного существеннее, чем ad-hoc-полиморфизм, повышают коэффициент повторного использования кода, поскольку определённая единственный раз функция реализует без дублирования заданное поведение для бесконечного множества вновь определяемых типов, удовлетворяющих требуемым в функции условиям. С другой стороны, временами возникает необходимость обеспечить различное поведение функции в зависимости от типа параметра, и тогда необходимым оказывается специальный полиморфизм.

Параметрический полиморфизмявляется синонимом абстракции типа . Он повсеместно используется в функциональном программировании, где он обычно обозначается просто как «полиморфизм».

В сообществе объектно-ориентированного программирования под термином «полиморфизм» обычно подразумевают наследование, а использование параметрического полиморфизма называют обобщённым программированием, или иногда «статическим полиморфизмом».

Объектно-ориентированные языки программирования

Многие современные языки специально созданы для облегчения объектно-ориентированного программирования. Однако следует отметить, что можно применять техники ООП и для не-объектно-ориентированного языка и наоборот, применение объектно-ориентированного языка вовсе не означает, что код автоматически становится объектно-ориентированным. Как правило, объектно-ориентированный язык (ООЯ) содержит следующий набор элементов:

Объявление классов с полями (данными — членами класса) и методами (функциями — членами класса).

Механизм расширения класса (наследования) — порождение нового класса от существующего с автоматическим включением всех особенностей реализации класса-предка в состав класса-потомка. Большинство ООЯ поддерживают только единичное наследование.

Полиморфные переменные и параметры функций (методов), позволяющие присваивать одной и той же переменной экземпляры различных классов.

Полиморфное поведение экземпляров классов за счёт использования виртуальных методов. В некоторых ООЯ все методы классов являются виртуальными.

Некоторые языки добавляют к указанному минимальному набору те или иные дополнительные средства. В их числе:

* Конструкторы, деструкторы, финализаторы;
* Свойства (аксессоры);
* Индексаторы;

Средства управления видимостью компонентов классов (интерфейсы или модификаторы доступа, такие как public, private, protected, feature и др.). Одни языки отвечают принципам ООП в полной мере — в них все основные элементы являются объектами, имеющими состояние и связанные методы. Примеры подобных языков — Smalltalk, Eiffel. Существуют гибридные языки, совмещающие объектную подсистему в целостном виде с подсистемами других парадигм как «два и более языка в одном», позволяющие совмещать в одной программе объектные модели с иными, и размывающие грань между объектно-ориентированной и другими парадигмами за счёт нестандартных возможностей, балансирующих между ООП и другими парадигмами (таких как множественная диспетчеризация, параметрические классы, возможность манипулировать методами классов как самостоятельными объектами, и др.). Примеры таких языков: CLOS, Dylan, OCaml, Python, Ruby, Objective-C. Однако, наиболее распространены языки, включающие средства эмуляции объектной модели поверх более традиционной императивной семантики. Алан Кэй назвал такие языки «склеиванием возможностей» (англ. agglutination of features) в противовес «чистоте стиля» (англ. crystalization of style) языков, воплощающих некую парадигму непосредственно[26]. Примеры таких языков — Симула, C++, Visual Basic, Delphi, Модула, Модула-2, Java, C#, PHP.