Лекция № 1.
Тема: **Введение в объектно-ориентированное программирование (ООП).**

1. Направления программирования

В программировании, как виде человеческой деятельности, выделяют основные направления для решения задачи преодоления сложности предметной области. Декомпозиция (разделение сложной задачи на более простые) - основной подход преодоления сложности во всех науках. При использовании декомпозиции задача делится на подзадачи на основании какого-либо критерия. В программировании в качестве критерия разделения выступают обычно:

1. Действия, выполняемые при решении задачи (этот критерий используют императивные (процедурные) языки - Fortran, Cobol, Algol, Pascal, Basic, Ada, C);
2. Данные и правила перебора вариантов, используемые при решении задачи (этот критерий используют логические языки - Prolog);
3. Действия и данные, используемые при решении задачи (этот критерий используют объектные и объектно-ориентированные языки - Smalltalk, C++, C#, Java, Ruby, Delphi);
4. Действия, представляемые как функциональные вычисления (этот критерий используют функциональные языки - Haskell, Lisp, F#);

Множество существующих языков программирования предназначены для формального описания решения разнообразных задач. Популярность конкретных языков программирования связана в основном с потребностью решения некоторого круга задач с помощью наиболее подходящего инструментария, которым выступает как сам язык программирования, так и среда разработки программ для данного языка. Так, рейтинг языков программирования на январь 2017 года ставит на 1-е место по популярности язык Java (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html):

Таблица 1- Рейтинг языков программирования.

Jan 2017	Jan 2016	Change	Programming Language	Ratings	Change
1	1		Java	17.278%	-4.19%
2	2		C	9.349%	-6.69%
3	3		C++	6.301%	-0.61%
4	4		C#	4.039%	-0.67%

Jan 2017	Jan 2016	Change	Programming Language	Ratings	Change
5	5		Python	3.465%	-0.39%
6	7	↑	Visual Basic .NET	2.960%	+0.38%
7	8	↑	JavaScript	2.850%	+0.29%
8	11	↑	Perl	2.750%	+0.91%
9	9		Assembly language	2.701%	+0.61%
10	6	⇊	PHP	2.564%	-0.14%
11	12	↑	Delphi/Object Pascal	2.561%	+0.78%
12	10	↓	Ruby	2.546%	+0.50%
13	54	⇈	Go	2.325%	+2.16%
14	14		Swift	1.932%	+0.57%
15	**13**	↓	**Visual Basic**	**1.912%**	**+0.23%**
16	19	↑	R	1.787%	+0.73%
17	26	⇈	Dart	1.720%	+0.95%
18	18		Objective-C	1.617%	+0.54%
19	15	⇊	MATLAB	1.578%	+0.35%
20	20		PL/SQL	1.539%	+0.52%

2. Кризис ПО

В 70-х годах 20 века понятие «кризис программного обеспечения» означал невозможность средствами процедурных языков решить задачу декомпозиции и объяснял причину неуспешности создания 70% программных проектов в те годы. Устранение причины кризиса было найдено в создании объектных и объектно-ориентированных языков. Такие языки вводили понятие класса, как группы объектов с близкими свойствами. Объект - совокупность данных и методов их обработки. Объект описывает некоторую физическую или логическую сущность. Например, класс «Студент» описывает группу объектов - студентов (физических лиц), задача которых - обучение по некоторым образовательным планам с целью получения некоторой квалификации. Пример данных объекта «Студент» - номер зачетной книжки, оценки по предметам, пример метода обработки данных - подсчет рейтинга студента. Такое разделение при автоматизации процесса учета студентов естественно, интуитивно понятно и ведет к более понятной логике создаваемого ПО, а значит к более быстрому решению с меньшими ошибками.

3. Недостатки процедурного программирования

По мере развития вычислительной техники возникали разные методики программирования. На каждом этапе создавался новый подход, который помогал программистам справляться с растущим усложнением программ. Первые программы создавались посредством ключевых переключателей на передней панели компьютера. Очевидно, что такой способ подходит только для очень небольших программ. Затем был изобретен язык ассемблера, который позволял писать более длинные программы. Следующий шаг был сделан в 1950 году, когда был создан первый язык высокого уровня Фортран.

Используя язык высокого уровня, программисты могли писать программы до нескольких тысяч строк длиной. Для того времени указанный подход к программированию был наиболее перспективным. Однако язык программирования, легко понимаемый в коротких программах, когда дело касалось больших программ, становился нечитабельным (и неуправляемым). Избавление от таких неструктурированных программ пришло после изобретения в 1960 году языков структурного программирования. К ним относятся языки Бейсик, Паскаль, C++ и многие другие. Структурное программирование подразумевает точно обозначенные управляющие структуры, программные блоки, отсутствие (или, по крайней мере, минимальное использование) инструкций GOTO, автономные подпрограммы, в которых поддерживается рекурсия и локальные переменные. Сутью структурного программирования является возможность разбиения программы на составляющие ее элементы.

Хотя структурное программирование, при его использовании для написания умеренно сложных программ, принесло выдающиеся результаты, даже оно оказывалось несостоятельным тогда, когда программа достигала

определенной длины (в несколько сотен тысяч строк). Чтобы написать более сложную программу, необходим был новый подход к программированию.

Процедурные языки предлагают деление всего процесса обработки информации в программе на участки (подпрограммы), которые реализуют некоторые действия, при этом данные, которые обрабатывают подпрограммы, могут быть заданы совсем в другой области определения. Разделение данных и подпрограмм их обработки приводит к ошибкам как на этапах создания ПО, так и на этапе эксплуатации. Повторное использование такого кода возможно или путем копирования кода, или путем создания библиотек. Защищенность данных при использовании процедурных языков низкая как на этапе создания, так и на этапе выполнения программы. Так как данные и подпрограммы связываются в процедурных языках на этапе компиляции, то гибкость программ, созданных с помощью процедурных языков, низкая. Недостатки процедурного программирования не позволяют создавать большие программные продукты коллективом разработчиков.

4. Достоинства ООП

ООП аккумулирует лучшие идеи, воплощенные в структурном программировании, и сочетает их с мощными новыми концепциями, которые позволяют оптимально организовывать ваши программы.

Объектно-ориентированное программирование позволяет вам разложить задачу на составные части. Каждая составляющая становится самостоятельным объектом, содержащим данные и коды их обработки. Такой механизм объединения данных с кодами называется ИНКАПСУЛЯЦИЕЙ. В языках, поддерживающих ООП, эта концепция реализуется с помощью классов.

Класс - это механизм для создания объектов. Объект представляет собой некоторую сущность. Возьмем, к примеру, телефон. Класс позволяет определять все атрибуты объекта. В нашем случае класс может содержать такие элементы данных, как номер и тип телефона (кнопочный или дисковый) и функции, работающие с телефоном, например dial, answer, и flash. Группируя данные об объекте и кодируя их в одной переменной, вы упрощаете процесс программирования и увеличиваете возможность повторного использования своего кода. Более понятным примером класса может служить проект постройки дома. Проект - это класс, а построенный по проекту дом - объект класса. Суть класса можно выразить одним словом - шаблон.

Поэтому, объявление класса является логической абстракцией, которая задает новый тип данных. Объявление же объекта создает физическую сущность объекта такого типа. То есть, объект занимает память, а задание типа - нет. Так же каждый объект класса имеет собственную копию всех переменных, объявленных внутри класса.

Класс, как независимый объект, можно НАСЛЕДОВАТЬ - то есть, на его основе создавать другой класс - производный - который может использовать свойства базового класса, добавляя к ним свои новые данные и методы (функции обработки данных). Таким образом создается ИЕРАРХИЯ КЛАССОВ.

И третья основа ООП - ПОЛИМОРФИЗМ (многоформенность) - разрешает использовать одно и то же имя для задания единого класса действий. Идея: один интерфейс - множество методов. Например, несколько функций могут иметь одно и то же имя, но выполнять различные действия. Функции различаются только количеством и типом параметров. Выбор же конкретного действия возлагается на программу, которая выбирает нужную функцию, исходя из анализа параметров. Это не значит, что в классе объявляются несколько функций с одним именем, но с различным количеством параметров - это обыкновенная перегрузка функции.

В классе некоторая функция, в будущем определяющая полиморфизм объекта класса, объявляется виртуальной. Затем создаются производные классы, переопределяющие данную виртуальную функцию для каждого конкретного случая. Конечно же, все эти функции имеют одно и то же имя. И изменение формы полиморфного объекта происходит при изменении "направления" указателя на объект производного класса. Указали на объект базового класса - одна форма объекта, указали на объект производного класса - другая форма объекта и так далее.

В зависимости от программы иногда не имеет смысла определять виртуальную функцию в базовом классе. Например, объекты производных типов могут настолько сильно отличаться, что им не нужно будет использовать виртуальный метод базового класса. В таком случае в базовом классе создается чисто виртуальная функция или функция без тела. Здесь каждый производный класс должен определить свою функцию вместо чисто виртуальной функции базового класса.

Концепция классов позволяет реализовать идею повторного использования программного кода. То есть однажды созданный класс можно использовать во многих программах. А при написании программы создаются объекты - переменные этого класса, которые обладают всеми возможностями класса (данными и методами). Таких переменных класса может быть множество.

Ошибочно считается, что программирование на языках типа Visual Basic, Delphi, C++ Builder и других, имеющих интерфейс типа "поводи мышью и создай объект" является объектно-ориентированным. Это не так. В подобных языках при написании программ создаются и используются объекты готовых классов. И это облегчает создание Windows-приложения.

Как Windows работает с программой? Вся работа Windows основана на сообщениях, которые она посылает программе в результате произошедшего события. Это может быть щелчок мышью, нажатие клавиши на клавиатуре и многое другое. Даже движение курсора по экрану - тоже событие. И программа должна отреагировать на сообщение (произошедшее событие) - выполнить какое-то действие.

Таким образом, программа управляется с помощью сообщений Windows. И программирование на Visual Basic, Delphi и других подобных языках - большей частью СОБЫТИЙНО-УПРАВЛЯЕМОЕ. ООП поддерживается этими языками, и даже C++, традиционно называемый объектно-ориентированным

языком, точнее было бы назвать языком, поддерживающим ООП.

Программирование только тогда становится объектно-ориентированным, когда начинается разбиение проблемы на отдельные объекты и разработка этих объектов как классов для многократного использования.

Итак, подведем итоги преимущества ООП:

1) Интуитивно понятное выделение в предметной области решаемой задачи объектов обработки; при этом защита данных и методов объектов может быть многоуровневой.
2) Повторное использование кода реализовано путем наследования, что обеспечивает гибкое распределение доступа к данным и методам.
3) Доступ к данным может меняться в зависимости как от самих данных, так и зависимости от условий использования данных.
4) ООП реализует разнообразные приемы эффективного создания больших проектов коллективом разработчиков.

5. Особенности языка Java

Хотя язык java в последние годы несколько теряет свои позиции, он по-прежнему остается одним из лучших для обучения основам объектно-ориентированного программирования.

Язык Java обладает следующими особенностями:

1) Java - современный язык программирования, отражающий все мировые тенденции в информационных технологий.
2) Особенностью языка является кроссплатформенность. Эффективность и корректность написанных на Java программ не зависит (или почти не зависит) от типа процессора или операционной системы.
3) Концепция языка логична и понятна.
4) Язык Java - строго типизированный язык, для которого проверки соответствия типов выполняются как на этапе компиляции, так и на этапе выполнения.
5) Язык Java реализует в полной мере все возможности ООП - инкапсуляцию, наследование, полиморфизм. Инкапсуляция - объединение данных и методов их обработки в специальных переменных типа класс (экземплярах класса - объектах) с целью разграничения доступа. Наследование - возможность создавать классы - потомки на базе классов - родителей (соответственно и объекты таких классов). Полиморфизм - возможность связывать данные и методы как во время компиляции программы, так и во время выполнения.
6) Современные среды создания программ на Java содержат все необходимые инструменты для реализации объектно- ориентированных технологий создания ПО.

6. Инструментальные средства Java Development Kit

Программы Java используют концепцию виртуальной Java-машины. Так, если обычно при компиляции программы (например, написанной на C++) на выходе мы получаем исполнительный машинный код, то в результате

компиляции Java-программы получают промежуточный байт-код, который выполняется не операционной системой, а виртуальной Java-машиной (Java Virtual Machine, JVM). Разумеется, предварительно виртуальная Java-машина должна быть установлена на компьютер пользователя. С одной стороны, это позволяет создавать достаточно универсальные программы (в том смысле, что они могут использоваться с разными операционными системами). Однако, с другой стороны, платой за такую «универсальность» является снижение скорости выполнения программ.

Для того чтобы программировать в Java, необходимо установить среды JDK (Java Development Kit — среда разработки Java) и JRE (Java Runtime Environment — среда выполнения Java).

JDK долгое время был базовым средством разработки приложений. Он не содержит никаких текстовых редакторов, а оперирует только с уже существующими java-файлами. Компилятор представлен утилитой javac (java compiler), виртуальная машина реализована программой java. Для тестовых запусков аплетов есть специальная утилита appletviewer.

Пакет Java Development Kit можно загрузить с web-страницы http://www.oracle.com/technetwork/java/javase/downloads. Способы инсталляции на разных платформах отличаются друг от друга.

После инсталляции пакета JSDK нужно добавить имя каталога jdk\bin в список путей, по которым операционная система может найти выполняемые файлы. Правильность установки пакета можно проверить, набрав команду java –version. На экране должно появиться, примерно, следующее:

```
java version "1.8.0_71"
Java(TM) SE Runtime Environment (build 1.8.0_71-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.71-b15, mixed mode)
```

6. Среда разработки программ

Для написания программ на языке Java достаточно использовать самый простой текстовый редактор, однако применение специализированных средств разработки (Eclipse, NetBeans, JetBrains IDEA, Java WorkShop, Java Studio и др.) предоставляет большой набор полезных и удобных функций. Существует несколько способов компиляции и запуска на выполнение программ, написанных на языке Java: из командной строки или из другой программы, например, интегрированной среды разработки. Для компиляции программы из командной строки необходимо вызвать компилятор, набрав команду javac и указав через пробел имена компилируемых файлов:

```
javac file1.java file2.java file3.java
```

При успешном выполнении этапа компиляции в директории с исходными кодами появятся файлы с расширением .class, которые являются java байт-кодом. Виртуальная Java-машина (JVM) интерпретирует байт-код и выполняет программу. Для запуска программы необходимо в JVM загрузить основной класс, т.е. класс, который содержит функцию main(String args[]).

Например, если в файле file1.java есть функция main(), которая располагается в классе file1, то для запуска программы после этапа компиляции

необходимо набрать следующее:

```
java file1
```

Компиляцию и запуск программ из интегрированных сред разработки необходимо осуществлять в соответствии с документацией на программный продукт.

7. Особенности консольного приложения.

Популярность операционных систем во многом обеспечивается доступным для разных категорий пользователей объектно-ориентированным графическим интерфейсом. Однако для многих программ, которые пишут программисты, графический интерфейс является избыточным. Для выполнения таких программ ОС поддерживают выполнение консольного приложения (без графического интерфейса пользователя).

Консольное приложение для Windows внешне выглядит подобно приложению MS - DOS, однако обладает и такими возможностями:

1) консольное приложение обеспечивает выполнение 32-х (64-х) разрядного приложения и выполняется в отдельной виртуальной машине;
2) консольному приложению доступны для использования многие Win API функции, например, возможно обеспечить работу с виртуальными драйверами Windows (обработка действий пользователя с мышью, клавиатурой, принтером и т.д.), работу с национальными шрифтами и т.д.
3) консольное приложение отличается компактностью кода и особенно эффективно в случае, если программа может реализовать функциональность без графического интерфейса.

8. Этапы разработки

Само по себе консольное приложение использует три вида функций для работы с консолью - ввод, вывод, а также управление параметрами консоли.

Для ввода данных используется класс Scanner из библиотеки пакетов Java. Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) — объектом System.out.

Пример простой программы “Hello, world!” выглядит следующим образом:

```
public class Test {
/**
* Основной метод, с которого начинается выполнение любой Java
программы.
```

```
*/
public static void main (String args[])
{
System.out.println("Hello, world!");
}

}
```

Для вывода какой-либо информации используются процедуры **System.out.print** и **System.out.println**. Последняя отличается от первой тем, что после вывода информации переходит в начало следующей строки.

Функция может принимать любое количество параметров, в том числе и нулевое.

Теперь выведем надпись "Hello, World!" с помощью IDE NetBeans:

```
package javaapplication1;
public class JavaApplication1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello, world!");

}
```

Запустив программу, увидим результат выполнения во встроенной консоли в нижней части рабочей области.

Теперь попробуем что-нибудь ввести с клавиатуры. Пусть это будет некое число, которое мы тут же выведем.

```
import java.util.Scanner; // импортируем класс
public class JavaApplication2 {
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);//создаём объект класса Scanner
  System.out.print("Введите целое число: ");
  int i = sc.nextInt(); // считываем целое число с потока ввода и
сохраняем в переменную
  System.out.println("Вы ввели: " + i);
 }
}
```

Таким образом можно создавать простые приложения, которые будут использовать для ввода и вывода консоль. Вот пример приложения, которое выводит таблицу значений синуса угла:

```
package javaapplication3;
import java.util.Scanner; // импортируем класс ввода данных
public class JavaApplication3 {
 public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     System.out.print("X = "); double X = sc.nextDouble();
     System.out.print("dX = "); double dX = sc.nextDouble();
     System.out.print("N = "); int N = sc.nextInt();
     System.out.println();
```

```
      System.out.println("    X     |    sin(X)");
      System.out.println("---------+------------");

   for(int i=0; i <= N; i++) {
     double A = X + dX * i;
     System.out.println( A+ " |"+Math.sin(A/180*Math.PI));
  }

  }
}
```

Функция System.out.println(), вызванная без параметров, просто переведет текущую позицию на начало новой строки, не выводя никаких данных.
Результат работы программы:

```
X = 90
dX = 10
N = 18
   X    |   sin(X)
---------+------------
90.0 |1.0
100.0 |0.984807753012208
110.0 |0.9396926207859084
120.0 |0.8660254037844387
130.0 |0.766044443118978
140.0 |0.6427876096865395
150.0 |0.49999999999999994
160.0 |0.3420201433256689
170.0 |0.17364817766693072
180.0 |1.2246467991473532E-16
190.0 |-0.17364817766693047
200.0 |-0.34202014332566866
210.0 |-0.5000000000000001
220.0 |-0.6427876096865393
230.0 |-0.7660444431189779
240.0 |-0.8660254037844385
250.0 |-0.9396926207859082
260.0 |-0.984807753012208
270.0 |-1.0
```