Прикладное программирование

Лекция №2.

1. Основные принципы ООП;
2. Типы данных;
3. Классы в Java;
4. Наследование в Java;

Что такое ООП?

Объектно-ориентированное программирование (ООП) - это методология программирования, опирающаяся на три базовых принципа:

- **Инкапсуляция** *(Encapsulation)*
- **Наследование** *(Inheritance)*
- **Полиморфизм** *(Polymorphism)*


А без ООП нельзя?
ЛЬЗЯ

Какую проблему решает ООП?

Понимание **концепции ООП** предоставляет один из самых эффективных подходов к современному программированию.

Раньше программисты использовали **функциональный** или **процедурный принцип программирования**.

Это значит, что все программы, большие и маленькие, писались в одном файле. И с течением времени такие программы становились всё больше и сложнее, что доставляло множество проблем при поддержке таких программ и внесении изменений.

Эту проблему и решает **объектно-ориентированное программирование**.

ООП позволяет **объединить данные и методы**, относящиеся к одной сущности, и работать с ними, **как с одним целым.**

В чем основная идея ООП?

При просмотре вакансий разработчиков любого уровня Вы часто можете встретить такое требование, как **«понимание ООП»**. В чем же суть этого требования?

ООП как стиль написания кода подразумевает построение **структуры программы**, которая состоит из взаимодействующих объектов.

Поэтому в рамках **ООП** полезно научиться **«мыслить объектами».**

Объекты расположены в строгой **иерархии**, **самостоятельны** и при этом определенным образом **взаимодействуют между собой.**

При этом Ваша программа состоит из модулей – **блоков**, которые решают строго поставленные задачи.

И изменения в этих участках кода никак не должны отражаться на прочих модулях Вашей программы.

Инкапсуляция

Инкапсуляция – это принцип, объединяющий **наборы данных** и **код**, который может управлять этими данными, а также обеспечивать защиту данных от прямого внешнего доступа и возможного неправильного использования.

Например, *если в классе могут быть реализованы внутренние вспомогательные методы или поля, к которым доступ для пользователя следует запретить, то тут и будет использоваться* ***инкапсуляция.***

В ООП **данные** называются **полями объекта**, **алгоритмы** - **объектными методами**, а **поля данных** и **методы** вместе – **членами класса**.

Для реализации **инкапсуляции** используются **модификаторы доступа.**

!!! Доступ к данным класса возможен только через методы того же класса.

Инкапсуляция

Другими словами, **инкапсуляция** позволяет **скрывать внутреннюю реализацию объекта.** Что же это значит?

Фактически, **инкапсуляция** – это механизм объединения данных в единый компонент, который дает возможность их спрятать и защитить. Представьте, будто Вы сами помещаете ваши данные в защитный кокон.

То есть Вы **ограничиваете доступ одних компонентов программы к другим.**

При этом **инкапсуляция** позволяет над каждой частью программы работать **изолированно**!

За счет этого Вы значительно повышаете надежность разрабатываемых программ, т.к. локализованные в объекте алгоритмы обмениваются с программой сравнительно небольшими объемами данных, а тип и количество этих данных обычно тщательно контролируются.

Модификаторы доступа

- **Public** - к переменной, методу или классу можно обращаться из любого места программы. Это **самая высокая степень открытости**, т.к. нет никаких ограничений.
- **Private** - к переменной, методу или классу можно обращаться только из того класса, где он объявлен. Для всех остальных классов этот метод или переменная – вне зоны видимости. Это **самая высокая степень защиты**. Такие методы не наследуются и не переопределяются. **Доступ к ним из класса-наследника также невозможен.**
- **Default (package-private)** – **модификатор по умолчанию** - если переменная или метод не помечены никаким другим модификатором. Переменные и методы видны всем классам пакета, в котором они объявлены, и только им.
- **Protected** – к переменной, методу или классу, помеченному модификатором **protected**, можно обращаться как **из его пакета (как с package-private)**, так и из всех **унаследованных от него классов.**

Наследование

Наследование – это принцип, который позволяет **создавать новые классы** на основе первоначальных, **добавляя новые методы и поля данных**.

Другими словами, **наследование** – это **передача свойств от одного объекта или класса другому**, который имеет еще и свои собственные свойства.

*Давайте проведем аналогию с **мобильными телефонами разных поколений**:*

- *Производители могут улучшить «**железо**»,*
- *Увеличить **экраны**,*
- *Усовершенствовать **камеры**,*
- *Увеличить **емкость** батареи,*
- *Изменить **материал** корпуса или сменить цветовую **палитру**,*
- *Обновить «**прошивку**»*
- *Или даже добавить **стилус**.*

Но в целом, наследуется все **те же основные свойства**, которыми мы уже давно пользуемся.

Наследование

Первоначальный класс называется **предком (родительским классом или суперклассом)**, а новые классы – его **потомками (наследниками или субклассами)**.

Объект-потомок автоматически наследует от родителя **все поля и методы**, а также может дополнять объекты новыми полями и даже «заменять» методы родителя или дополнять их (переопределение).

Множество классов, связанных отношением наследования, называется **иерархией классов**, а класс, который состоит во главе иерархии – **базовым**.

Если рассматривается поведение объектов программы, характерное для объектов **класса А и его потомков**, то говорят **об иерархии, которая начинается с класса А.**

Наследование бывает **одиночное** и **множественное** (**в Java запрещено!**).

В Java все классы являются **потомками класса Object!**

Полиморфизм

Полиморфизм – это способность объектов с одним интерфейсом иметь различную реализацию.

Другими словами, **полиморфизм** – **один интерфейс, множество реализаций**.

Целью полиморфизма является использование одного имени при выполнении действий общих **для родительского класса** и его **подклассов.**

Например,

- *У нас есть три класса: **Круг**, **Квадрат** и **Треугольник**.*
- *У всех классов один класс-родитель: **Фигура**.*
- *У всех классов есть метод **getSquare()**, который считает и возвращает площадь.*
- *У всех фигур площадь вычисляется по-разному, а так же свой набор входных параметров.*

Следовательно, реализация одного и того же метода – различная.

Абстракция

Абстракция – это выделение и представление существенных отличительных признаков, свойств, характеристик в терминах программирования.

Абстракция позволяет выделять из некоторой сущности только необходимые характеристики и методы, которые в полной мере описывают объект в рамках поставленной задачи.

Например, *мы создаем класс* ***для описания студента****.*

Мы выделяем только некоторые характеристики:

- *ФИО,*
- *Курс,*
- *Группа,*
- *Номер зачетной книжки.*

При этом абсолютно не важно, где и как данный класс расположен в иерархии классов.

Классы и объекты

- **Класс** – это описание того, как будет устроен объект, являющийся экземпляром данного класса, а также какие методы объект может вызывать. Так как экземплярами классов являются объекты, то классы называют объектными типами.

- **Объект** – конкретный представитель одного из известных классов. Все объекты, являющиеся экземплярами некоторого класса, имеют одинаковые наборы полей данных (атрибуты объекта), но с независимыми значениями этих данных для каждого объекта. Значения полей данных объекта задают его состояние, а методы – его поведение. Сами объекты безымянны, и доступ к ним осуществляется только через объектные переменные.


Java.
Туда и обратно...

Алфавит языка и числовые константы в Java

- **Алфавит** языка Java состоит из букв, десятичных цифр и специальных символов.

- Буквами считаются **латинские буквы** (кодируются в стандарте ASCII), **буквы национальных алфавитов** (кодируются в стандарте Unicode), а также соответствующие им символы, кодируемые управляющими последовательностями.

- В программах разрешается пользоваться десятичными и шестнадцатеричными целыми числовыми константами. Шестнадцатеричная константа начинается с символов **0x** или **0X**, после чего идет само число в шестнадцатеричной нотации.

Алфавит языка и числовые константы в Java

- Язык Java является **регистро-чувствительным**. Исходные коды программ Java набираются в виде последовательности символов Unicode.

- **Управляющая последовательность** применяется в случае, когда требуется использовать символ, который обычным образом в текст программы ввести нельзя. Простая управляющая последовательность начинается с символа “\”, после которого идѐт управляющий символ. Управляющая последовательность для кодирования символа Unicode начинается с последовательности из двух символов -“\u”, после которой следует четыре цифры номера символа в шестнадцатеричной нотации. Например, **\u1234**.

Алфавит языка и числовые константы в Java

- **Идентификаторы** – это имена переменных, процедур, функций и т.д. В идентификаторах можно применять только буквы и цифры, причем первой всегда должна быть буква, а далее может идти произвольная комбинация букв и цифр. Длина идентификатора в Java любая.
- **Переменная** – это именованная ячейка памяти, содержимое которой может изменяться. При объявлении переменной сначала указывается тип переменной, а затем идентификатор задаваемой переменной.
- Типы в Java делятся на **примитивные** и **ссылочные**. Существует несколько предопределенных примитивных типов, все остальные – ссылочные. Все пользовательские типы кроме типов-перечислений являются ссылочными. Значение **null** соответствует ссылочной переменной, которой не назначен адрес ячейки с данными.

Ссылочные переменные

В каждой переменной **примитивного типа** содержится свое значение, и его изменение не влияет на значения других переменных. Имя переменной примитивного типа можно рассматривать как единственное и неизменяемое по ходу работы программы имя ячейки с данными.

Переменные ссылочного типа содержат адреса данных, а не сами данные. Присваивания для таких переменных меняют адреса, но не данные, а, кроме того, для них выделяется одинаковое количество памяти независимо от типа объектов, на которые они ссылаются.

В Java ссылочные переменные используются для работы с объектами: в этом языке программирования используется **динамическая объектная модель**, и все объекты создаются динамически, с явным указанием в программе момента их создания.

Это отличает Java от C++, языка со статической объектной моделью, в котором могут существовать как статически заданные объекты, так и динамически создаваемые.

Динамическая и статическая объектные модели

C++

```
class ClassA {
  private int i;
  public void whoAmI() {
    printf("I am class A");
  }
}
ClassA myClass1;
ClassA * myClass2 = new ClassA();
```

Java

```
 class ClassA {
  private int i;
  public void whoAmI() {
    System.out.println("I am class A");
  }
}
ClassA myClass2 = new ClassA();
```

Простые специальные символы языка

+	Оператор сложения
–	Оператор вычитания
*	Оператор умножения
/	Оператор деления
%	Оператор остатка от целочисленного деления
=	Оператор присваивания
~	Оператор побитового дополнения (побитовое “не”)
?	Вопросительный знак – часть тернарного (состоящего из трёх частей) условного оператора “? :”
:	Двоеточие – часть условного оператора “? :”. Также используется для задания метки – ставится после имени метки
^	Оператор “исключающее или” (XOR)
&	Оператор “побитовое и” (AND)
|	Оператор “побитовое или” (OR)
!	Оператор “НЕ”
>	Больше
<	Меньше
{	Левая фигурная скобка – открытие блока кода
}	Правая фигурная скобка – закрытие блока кода

Простые специальные символы языка

,	Запятая - разделитель в списке параметров оператора; разделитель в составном операторе
.	Точка – десятичный разделитель в числовом литерном выражении; разделитель в составном имени для доступа к элементу пакета, класса, объекта, интерфейса
(	Левая круглая скобка – используется для открытия списка параметров в операторах и для открытия группируемой части в выражениях
)	Правая круглая скобка – используется для закрытия списка параметров в операторах и для закрытия группируемой части в выражениях
[	Левая квадратная скобка – открытие индекса массива
]	Правая квадратная скобка – закрытие индекса массива
;	Точка с запятой – окончание оператора
'	Апостроф (одиночная кавычка) – открытие и закрытие символа
”	Двойные кавычки – открытие и закрытие строки символов
\	обратная косая черта (backslash) – используется для задания управляющих последовательностей символов
	знак пробела (невидимый)
	знак табуляции (невидимый)
@	Коммерческое а (“эт”) – знак начала метаданных

Составные специальные символы языка

++	Оператор инкремента (увеличения на 1); x++ эквивалентно x=x+1
--	Оператор декремента (уменьшения на 1); x-- эквивалентно x=x-1
&&	Оператор “логическое И” (AND)
||	Оператор “ логическое ИЛИ” (OR)
<<	Оператор левого побитового сдвига
>>>	Оператор беззнакового правого побитового сдвига
>>	Оператор правого побитового сдвига с сохранением знака отрицательного числа
==	Равно
! =	не равно
+=	y+=x эквивалентно y=y+x
-=	y-=x эквивалентно y=y-x
*=	y*=x эквивалентно y=y*x
/=	y/=x эквивалентно y=y/x
%=	y%=x эквивалентно y=y%x
|=	y|=x эквивалентно y=y|x
^=	y^=x эквивалентно y=y^x

Составные специальные символы языка

>>=	y>>=x эквивалентно y= y>>x
>>>=	y>>>=x эквивалентно y= y>>>x
<<=	y<<=x эквивалентно y= y<<x
/*	Начало многострочного комментария.
/**	Начало многострочного комментария, предназначенного для автоматического создания документации по классу.
*/	Конец многострочного комментария (открываемого как /* или /**).
//	Однострочный комментарий

Ключевые слова языка Java

abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const*	float	native	super	while

Ключевые слова языка Java

abstract	определяет абстрактный класс или метод
boolean	логический тип (1 байт)
break	завершает обработку цикла или оператора switch
byte	8-битный целый тип
case	указывает на возможность выбора в операторе switch
catch	перехватывает исключения
char	символьный тип (2 байта)
class	определяет тип класса
const*	зарезервировано, но не используется

Ключевые слова языка Java

continue	продолжает выполнение цикла со следующей итерации
default	указывает на значение по умолчанию в операторе при отсутствии совпадений
do	начинает цикл do/while
double	вещественное число с двойной точностью (8 байт)
else	указывает на альтернативные действия оператора if
enum	определяет перечислимый тип
extends	определяет предка расширяемого класса
final	указывает, что переменная не может меняться, класс не может порождать наследников и метод не может быть переопределен
finally	указывает часть блока try, выполняемого всегда
float	вещественное число с одинарной точностью (4 байта)

Ключевые слова языка Java

for	начинает цикл for
if	условный оператор
implements	указывает, что класс реализует (определяет методы) интерфейс
import	указывает пакеты (библиотеки классов) интерфейса
instanceof	возвращает истину, если указанный объект является экземпляром класса
int	целочисленный тип (4 байта)
interface	опеделяет абстрактный тип с методами, которые далее может реализовать класс
long	длинный целочисленный тип (8 байт)
native	указывает, что метод реализован в C или каким-то другим машиннозависимым способом

Ключевые слова языка Java

new	распределяет новый объект или массив
package	совокупность классов (библиотека)
private	модификатор, указывающий, что только метод данного класса может получить доступ к объекту
protected	модификатор, указывающий, что только метод данного класса, его подклассов и других классов пакета может получить доступ к объекту
public	модификатор, указывающий, что методы всех классов могут получить доступ к объекту
return	возвращает управление из функции вызывавшему процессу
short	короткий целочисленный тип (2 байта)
static	модификатор, указывающий на структуру объекта
super	обращение к суперклассу объекта или конструктора

Ключевые слова языка Java

switch	используется с одним или более операторов case для создания условного оператора
transient	указывает, что данная переменная не должна сохраняться при сериализации объекта
try	определяет блок кода, где может возникнуть исключение
void	указывает, что метод не возвращает значения
volatile	указывает, что переменная изменяется асинхронно и компилятор не должен пытаться оптимизировать ее использование
while	определяет начало цикла while

Типы данных в Java

Примитивный тип	Тип-оболочка	Назначение
boolean	**Boolean**	логическое значение: true или false (1 байтовое)
char	**Character**	16-битовое (2 байтовое) символьное значение в кодировке Unicode
byte	**Byte**	8-битовое (1 байтовое) целое число со знаком
short	**Short**	16-битовое (2 байтовое) короткое целое число со знаком
int	**Integer**	32-битовое (4 байтовое) обычное целое число со знаком
long	**Long**	64-битовое (8 байтовое) длинное целое число со знаком
float	**Float**	32-битовое (4 байтовое) вещественное число
double	**Double**	64-битовое (8 байтовое) вещественное число

Преобразование из строкового представления с помощью типов-оболочек

Статические методы классов-оболочек предоставляют возможность преобразования строкового представления чисел в значения соответствующих типов:

- *Byte.parseByte(строка), Short.parseShort(строка), Integer.parseInt(строка), Long.parseLong(строка), Float.parseFloat(строка), Double.parseDouble(строка)*
  преобразуют строку в число соответствующего типа.

- *Byte.valueOf(строка), Short.valueOf(строка), Integer.valueOf(строка), Long.valueOf (строка), Float.valueOf (строка), Double.valueOf (строка)*
  аналогичны им, но **возвращают не числовые значения, а объекты соответствующих типов-оболочек.**

Определение констант в Java

Константы в Java определяются добавлением ключевого слова **final**.

Оно сообщает о том, что после присвоения значения переменной это значение остается неизменным.

Принято имена констант писать в верхнем регистре:

```
final int DOTS_PER_INCH = 300;
```

Ключевое слово **const** в Java не используется, вместо него следует использовать **final**.

Управляющие структуры в Java

Конструкции управления потоком управления в Java аналогичны конструкциям языков C и C++ с двумя исключениями:

- Отсутствует оператор **goto**;
- Существуют обладающие метками версии операторов **break** и **continue**, позволяющая прерывать выполнение вложенных циклов.

Пример:

```
label_for1:
        for(int i=1; i<=20; i++) {
                for(int j=1; j<=20; j++) {
                        if(i*j == (i*j/2)*2)
                                continue label_for1;
                        System.out.println("...");
                };
        };
```

Пример программы на Java

```
// Задание пакета, которому принадлежит класс
package bsu.rfct.java.group1.lab0.Novoseltsev.varA0;

public class Lab1 { // объявление главного класса Lab1 приложения
  public Lab1() {
        // конструктор класса Lab1 вызывается из метода main
        // в простых программах может не определяться
}
public static void main(String[] args) {
      // main() – главный метод, автоматически вызывается при запуске,
      // обязательно должен быть определен как public static void.
      // Из него вызываются и управляются все классы и объекты приложения,
      // параметры командной строки находятся в массиве строк args,
      // после выполнения метода main, приложение завершает свою работу
      System.out.println("Лабораторная работа 1");
  }
}
```

Логическая структура приложения

- Для логического группирования множеств классов в семантически-связанные группы в Java применяется понятие **пакета** (package). Пакеты обеспечивают независимые пространства имен (namespaces), а также ограничение доступа к классам.

- Классы всегда задаются в каком-либо пакете. Пакеты могут быть вложенными с произвольным уровнем вложения. Для того чтобы поместить класс в пакет, требуется поместить исходный код класса в папку, соответствующую необходимому пакету, и объявить имя пакета в начале файла с исходным кодом класса:

```
package имя_пакета;
```

Логическая структура приложения

- Если декларация имени пакета отсутствует, считается, что класс принадлежит пакету с именем ***default***.

- Вложенным пакетам соответствуют составные имена. Например, если имеется пакет с именем ***bsu***, в который вложен пакет с именем ***rfct***, в который вложен пакет с именем ***java***, то объявление, что класс с именем **MyClass1** находится в пакете ***bsu.rfct.java***, выглядит так:

```
package bsu.rfct.java;
class MyClass1 { … }
```

Хранение пакетов и классов на диске

- Каждому пакету в файловой системе соответствует папка с исходными кодами классов, имя которой совпадает с именем пакета. Вложенным пакетам соответствуют вложенные папки, т.е. если пакет ***rfct*** вложен в пакет ***bsu***, то папка ***rfct*** будет находиться внутри папки ***bsu.***
- Хотя в качестве разделителя имен пакетов в программе используется точка (независимо от типа операционной системы), в различных операционных системах вложенность папок будет обозначаться поразному, например: в ***MS Windows*** – ***bsu\rfct***, в ***Unix*** – ***bsu/rfct/***, в ***Mac OS*** – ***bsu:rfct:***
- В большинстве случаев* каждому классу Java соответствует файл на диске, в который включаются и определение, и реализация класса. Файлы размещаются по папкам в соответствии с принадлежностью классов пакетам.

Использование классов из других пакетов

- Класс может использовать общедоступные (**public**) классы из других пакетов напрямую, с указанием полного имени класса в пространстве имен, включающего имя пакета. Например, доступ к классу **MyClass2** осуществляется как ***bsu.rfct.java.MyClass2***.
- Для доступа к именам из другого пакета «напрямую», без указания каждый раз полного пути в пространстве имен, необходимо сначала выполнить импорт отдельного класса пакета или всего пакета целиком в текущее пространство имен. Это делается с помощью ключевого слова ***import*** (расположенного непосредственно за объявлением имени текущего пакета с помощью ***package***) с указанием имени импортируемого класса или пакета, например:

```
package bsu.rfct.java;
import bsu.fami.java.HelpFromAFriend;
```

- В дальнейшем класс **HelpFromAFriend** можно использовать напрямую, без указания перед ним имени пакета ***bsu.fami.java.***

Импорт всех классов из пакетов

- Если необходимо импортировать всех классы пакета, то в блоке ***import*** после имени пакета вместо имени класса следует написать * (символ звездочки), например:

```
import bsu.rfe.java.*;
```

- **PS**: импортируются только те классы, которые находятся непосредственно в указанном пакете. Импорта классов из вложенных пакетов не происходит.
- **PPS**: все классы ядра языка Java, содержащиеся в пакете ***java.lang***, импортируются автоматически без указания имени пакета.

Объявление и реализация класса

Определение классов в Java осуществляется следующим образом:

```
[<Модификаторы_класса>] class <Имя_класса>
        [extends <Имя_суперкласса>]
        [implements <Интерфейс_1>, <Интерфейс_2>, ...] {
// Описания методов и переменных экземпляров класса
}
```

Модификаторы класса, которые можно использовать:

- **public** – модификатор, задающий публичный (общедоступный) уровень видимости. Если он отсутствует, действует пакетный уровень доступа - класс доступен только элементам того же пакета.
- **abstract** – модификатор, указывающий, что класс является абстрактным, то есть у него не бывает экземпляров (объектов). Обязательно объявлять класс абстрактным в случае, если какой-либо метод объявлен как абстрактный.
- **final** – модификатор, указывающий, что класс является окончательным, то есть что у него не может быть потомков.

Объявление и реализация класса

Определение классов в Java осуществляется следующим образом:

```
[<Модификаторы_класса>] class <Имя_класса>
        [extends <Имя_суперкласса>]
        [implements <Интерфейс_1>, <Интерфейс_2>, ...] {
// Описания методов и переменных экземпляров класса
}
```

Секция [extends <superclass_name>] используется при определении данного класса потомком некоторого суперкласса (фактически заменяет оператор наследования C++ «:»).

Секция [implements <interface_1>, <interface_2>, ...] появляется в определении класса, если он реализует один или несколько интерфейсов.

Объявление набора полей данных класса

Набор данных класса состоит из множества полей, включающих:

- **Переменные** (поля) **экземпляра класса** (Java-объекта) содержат его данные, значения этих переменных определяют состояние конкретного экземпляра данного класса.
- **Переменные класса**, являющиеся общими для всех объектов класса и объявляемые с модификатором **static**.

Модификаторы **public**, **private**, **protected** предназначены для управления доступом к полям данных класса. Если модификатор доступа явно не указан, то поле имеет **пакетный тип доступа.**

Модификатор **final** помечает окончательное поле, которому должно быть присвоено первоначальное значение, после чего новое значение ему уже не может быть присвоено. Значения констант класса объявляются с помощью модификаторов **static** и **final**.

```
[public|protected|private] [static] [final]
  <Тип_поля> <Имя_поля> [ = <Начальное значение> |
      = <Имя_конструктора>(<Аргументы_конструктора>)]
```

Объявление конструкторов класса

Объекты создаются с помощью **конструктора** – специальной подпрограммы, занимающейся созданием объекта и инициализацией его полей.

Для конструктора **не указывается тип возвращаемого значения**, он не является ни методом объекта, ни методом класса (в конструкторе через ссылку **this** доступен сам объект и его поля). Конструктор в сочетании с ключевым словом **new** возвращает ссылку на создаваемый объект и может считаться особым видом методов, соединяющим в себе черты методов класса и методов объекта.

Если в объекте при создании не нужна дополнительная инициализация, можно использовать конструктор без параметров (по умолчанию), присутствующий для каждого класса.

Первым оператором в конструкторе должен быть вызов конструктора суперкласса с помощью ключевого слова **super**, после которого идет необходимый для конструктора предка список параметров.

Для обращения к другой (перегруженной) версии конструктора с другим набором входных параметров, в конструкторе часто используют ключевое слово **this**, после которого идет необходимый для список параметров (обращение к **this** должно быть первым оператором).

Порядок вызова конструкторов в иерархии объектов

Порядок вызовов при создании объекта случайного дочернего класса:

1. Создается объект, в котором все поля данных имеют значения по умолчанию (нули или **null**).
2. Вызывается конструктор дочернего класса.
    1. Конструктор дочернего класса вызывает конструктор непосредственного предка, а также по цепочке все конструкторы предков в иерархии наследования в этих классах вплоть до класса **Object**.
    2. Проводится инициализация полей родительской части объекта значениями, заданными в декларации класса-предка.
    3. Выполняется тело конструктора класса-предка.

3. Проводится инициализация полей дочерней части объекта значениями, заданными в декларации дочернего класса.

4. Выполняется тело конструктора дочернего класса.

Порядок вызова конструкторов в иерархии объектов

Кроме конструкторов для инициализации полей в Java могут применяться блоки инициализации класса и объекта:

```
[<Модификаторы>] class <Имя_класса> [extends <Имя_суперкласса>] {
   // Задание полей
   static {
      // тело блока инициализации класса
   }
   {
      // тело блока инициализации объекта
   }
   // Задание методов класса, методов объекта, конструкторов
}
```

Блоков инициализации класса и объекта может быть несколько.

Порядок выполнения при наличии блоков инициализации следующий:

1. Инициализация полей данных и выполнение блоков инициализации класса (в порядке записи в декларации класса);
2. Выполнение блоков инициализации объекта;
3. Выполнение тела конструктора класса.

Задание методов класса

Методы определяют поведение объектов класса.

Описание метода состоит из двух частей:

- **сигнатуры**, определяющей его модификаторы, имя, число и типы параметров метода
- **тела метода**, в котором описываются выполняемые действия.

```
[<Модификаторы>] <Тип_возвращаемого_значения>
<Имя_метода>(<Аргументы_метода>) {
    // Тело метода
}
```

Модификаторы – это зарезервированные слова, определяющие:

- Правила доступа к методу (***private***, ***protected***, ***public***). Если модификатор не задан, по умолчанию применяется пакетный доступ.
- Принадлежность к общим методам класса (***static***).
- Невозможность переопределения метода в потомках (***final***).
- Способ реализации (***native*** – заданный во внешней библиотеке DLL, написанной на другом языке программирования; ***abstract*** – абстрактный, не имеющий реализации).
- Синхронизацию при работе с потоками (***synchronized***).

Передача и изменение аргументов методов класса

В Java применяется передача аргументов метода **по значению.**

Это обуславливает ряд особенностей при работе с входными аргументами метода в Java:

- метод не может изменить значение параметра примитивного типа (т.е. числа или булевского значения);
- метод не может изменить параметр ссылочного типа так, чтобы он ссылался на другой объект;
- метод может изменить внутреннее значение параметра ссылочного (т.е. объектного) типа с помощью предложенных тем методов.

C++ позволяет передавать аргументы по значению и по ссылке.

Так как C++ позволяет работать с указателями, то при передаче по значению можно использовать в качестве параметров указатели на области памяти, где располагаются данные, и изменять их с помощью операции разыменования.

Java позволяет передавать аргументы только по значению. Как мы помним, указатели в Java не допускаются.

На сегодня все!

Увидимся через 2 недели – 5 октября.

На следующей лекции мы поговорим об объектных возможностях Java, наследовании, интерфейсах и некоторых стандартных классах.