# Лекция 2. Типизация в Java

# Занятие 2

**Типизация** *– это способ определить свойства и поведение объекта и защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.*

# Специфика ссылочных типов

Переменной ссылочного типа возвращается ***ссылка (адрес) на значение*** в памяти, поэтому говорят, что переменная ***ссылочного типа хранит ссылку на значение, а не само значение (в отличие от простых типов)***.

Ссылочные типы в Java – это классы и массивы. Имена ссылочных типов пишуться с заглавной буквы.

Переменные ссылочного типа инициализируются с помощью оператора ***new***:

Object obj = new Object(); // класс Object из библиотеки Java (1)

Когда создается перменная ссылочного типа, в памяти выделяется область для создаваемого объекта, в которой хранятся все параметры и значения, а самой переменной присваивается ссылка на этот объект (т.е. фактически переменной ссылочного типа данных возвращается некий адрес в памяти, обратившись по которому, можно получить состояние данной перменной).

Строка (1) – это ***выражение***, состоящие из двух частей – левой и правой. Части связаны друг с другом с помощью оператора присваивания («=»). Результат вычисления в правой части выражения **присваивается** переменной, объявленной в правой части выражения.

Согласно определению выражения: присвоить переменной obj ***ссылку*** на созданный в памяти типа Object. В случае с примитивными типами данных, переменным возращалось само инициализированное значание. Эта принципиальная разница становится очевидной при присваивании значений одних перменных другим.

Рассмотрим 2 случая:

int i =5; //проинициализировали перменную значением 5

int j = i; //проинициализировали переменную j таким же значением, что и переменная i.

j = 2\*j; //увеличили значение переменной j, при это значение i осталось 5.

В случае с объектами перменной присваивается ссылка на объект в памяти, т.е. перменных в памяти будет несколько, но они все будут иметь ссылку на один и тот же объект.

Object obj1 = new Object(); // создали новый объект и вернули obj1 ссылку на него.

Object obj2 = obj1; //создали новую перменную obj2 и вернули ей ссылку на тот же объект, на который ссылается переменная obj1.

Для ссылочных типов так же определена операция сравнения «=», но в этом случае, сравничваются не сами значения (объекты в памяти), а ссылки на них. Например:

Object obj1 = new Object(); // 1

Object obj2 = obj1; //2

If(obj1 == obj2) // 3

{

System.out.print(“it’s the same object, but different variables !!!”);

}else{

System.out.print(“it’s the deferent objects in memory !!!”);

}

Рассмотрим приведенный выше код:

1. В строке 1 происходит объявление и инициализация переменной obj1 типа Object. Для инициалиации перменной используется оператор new, значит, это перменная ссылочного типа, и после ее создания в памяти, будет возвращена ссылка (адрес) на этот объект. Ссылка будет присвоена перемененной obj1 типа Object.
2. В строке 2 происходит объявление и инициализация перменной obj2 типа Object. Она инициализируется значением перменнной obj1 типа Object, которая хранит адрес на объект, созданный на шаге 1. Т.е. переменной obj2 присваивается тот же адрес, который хранит переменная obj1.
3. Проверка условия равенства значений перменных – равно ли значение перменной obj1 значению переменной obj2? Учитывая, что они хранят одну и ту же ссылку – да.

Изменим код в строке 2 на: Object obj2 = new Object();.

Рассмотрим преобразованный код:

1. То же самое, что и в предыдущем примере.
2. В строке 2 происходит объявление и инициализация переменной obj2 типа Object. Для инициалиации перменной используется оператор new, значит, это перменная ссылочного типа, и после ее создания в памяти, будет возвращена ссылка (адрес) на этот объект. Ссылка будет присвоена перемененной obj2 типа Object.
3. Проверка условия равенства значений переменных – равно ли значение переменной obj1 значению переменной obj2? Учитывая, что они хранят ссылки на разные объекты (было создано 2 разных объекта, так как произошло 2 операции new) – нет.

# Массивы

Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по индексу.

Размер (длина массива) — это количество элементов в массиве. Размер массива задаётся при создании массива и не может быть изменён в дальнейшем, т. е. нельзя убрать элементы из массива или добавить их туда, но можно существующим элементам присвоить новые значения.

Индекс начального элемента — 0, следующего за ним — 1 и т. д. Индекс последнего элемента в массиве — на единицу меньше, чем размер массива (см. рис. 1).

|  |
| --- |
| C:\Users\Daria\Desktop\array.png |
| Рис. 1. Схема массива |

В Java массивы являются **объектами**. Это значит, что переменная, объявленная, как массив, хранит ссылку. Но какую ссылку, если массив – это множество объектов одного типа? Ссылку на первый элемент массива. Таким образом, индекс каждого элемента – это число, на которое надо сместить указатель (ссылку), чтобы получить следующий элемент массива.

Чтобы создать массив надо объявить для него подходящее имя, а затем с этим именем связать нужный фрагмент памяти, где и будут друг за другом храниться значения элементов массива. Возможные следующие варианты объявления массива:

тип[] имя; // знак массива стоит после объявления типа  
тип  имя[]; // знак массива стоит после имени переменной

Где тип — это тип элементов массива, а имя — уникальный (незанятый другими переменными или объектами в этой части программы) идентификатор, начинающийся с буквы.

Примеры:

int[] a;  
double[] ar1;  
double  ar2[];

В примере объявлено три массива. С первом именем **a** сможет быть далее связан массив из элементов типа int, а с именами ar1 и ar2 далее смогут быть связаны массивы из вещественных чисел (типа double).

**На этапе объявления массива под его элемента память не выделяется.**

Инициализация массива:

a = new int[10]; // массив  из 10 элементов типа int  
int n = 5;  
ar1 =  new double[n]; // Массив из 5 элементов double  
ar2 = {3.14, 2.71, 0, -2.5, 99.123}; // Массив из 6 элементов типа double

При инициализации указывается размер массива, либо сразу перечислить через запятую все желаемые элементы в фигурных скобках (при этом размер будет вычислен автоматически на основе той последовательности элементов, которая будет указана).

Если массив был создан с помощью оператора **new**, то каждый его элемент получает значение по умолчанию. Каким оно будет определяется на основании типа данных (0 для int, 0.0 для double и т. д.).

Объявить имя для массива и создать сам массив можно было на одной строке по следующей схеме:

тип[] имя  = new тип[размер];  
тип[] имя = {эл0, эл1, …, элN};

Примеры:

int[] mas1 = {10,20,30};  
int[] mas2  = new int[3];

Чтобы получить к элементу массива, нужно указать имя массива и за ним индекс элемента в квадратных скобках. Элемент массива с конкретным индексом ведёт себя также, как переменная. Например, чтобы вывести последний элемент массива mas1 мы должны написать в программе:

System.out.println("Последний  элемент массива " + mas1[2]);

Чтобы задать значение элементу массива, необходимо обратиться по индексу к массиву, и с помощью оператора присваивания задать значение выбранного элемента:

mas2[0] = 10;

mas2[1] = 20;

mas2[2] = 30;

Длину любого созданного массива не обязательно запоминать, потому что имеется свойство, которое его хранит. Обратиться к этому свойству можно дописав **.length** к имени массива. Например:

int razmer = mas1.length;

Это неизменяемое программистом значение (т. е. ему нельзя ничего присваивать), свойство length можно только читать. Используя это свойство можно писать программный код для обработки массива даже не зная его конкретного размера.

# Практическая часть

# Управляющие операторы в Java. Операторы циклов

**Неопределенные циклы**

Существует два вида повторяющихся циклов. В случае, когда заранее невозможно предсказать количество итераций цикла неопределенный цикл - **while**, выполняет тело цикла, только пока выполняется его условие. Вот его общий вид:

**while (условие){**

**/\*дейсвтия\*/**

**}**

Условие цикла **while** проверяется в самом начале, при каждой итерации. Следовательно, возможна ситуация, когда код, содержащийся в блоке, не будет выполнен никогда. Если вы хотите, чтобы блок выполнялся хотя бы один раз, проверку условия нужно перенести в конец. Это можно сделать с помощью цикла **do/while**. Его синтаксис выглядит так:

**do{**

**/\*дейсвтия\*/**

**} while (условие);**

Этот оператор выполняет блок действий и только **затем** проверяет условие. После он повторяет выполнение блока и вновь проверяет условие и т.д.

**Определенные циклы**

Цикл **for** — самая распространенная конструкция для выполнения повторяющихся действий, количество которых контролируется счетчиком или аналогичной переменной, обновляемой на каждой итерации. Общий вид:

**for(инициализация; условие; повторение){**

**/\*дейсвтия\*/  
}**

**Первый** элемент цикла обычно выполняет инициализацию счетчика, **второй** формулирует условие выполнения тела цикла, а **третий** определяет способ обновления счетчика. Элементами оператора цикла **for** могут быть практически ***любые*** операторы, существуют неписанные правила хорошего тона, которые утверждают, что три элемента оператора **for** должны только инициализировать, проверять и обновлять один и тот же счетчик. Если не придерживаться этих правил, можно написать совершенно невразумительные циклы. Даже следуя правилам хорошего тона, можно сделать многое. Например, можно написать цикл с убывающим счетчиком.

Данный вид цикла может иметь несколько вариаций задания:

1. Бесконечный цикл: **for(;;){}**
2. Цикл с условием выхода в виде буленовского типа: **for(int a=0; !done; a++){}**
3. Цикл с меняющимися параметрами:

int a, b;

**for(a=0, b=2; a<=b; a++,b--){}**

Используя циклы, можно «обойти» весь массив. Например, можно вывести на экран элементы любого массива с именем ar2:

for(int i = 0; i <= ar2.length  - 1; i++) {  
  System.out.print(ar2[i] + "  ");  
}

Для краткости удобнее менять нестрогое неравенство на строгое, тогда не нужно будет вычитать единицу из размера массива. Давайте заполним массив целыми числами от 0 до 9 и выведем его на экран:

for(int i = 0; i <  ar1.length; i++) {ar1[i] =  Math.floor(Math.random() \* 10);  
  System.out.print(ar1[i] + "  ");  
}

Обратите внимание, на каждом шаге цикла мы сначала отправляли случайное значение в элемент массива с i-ым индексом, а потом этот же элемент выводили на экран. Но два процесса (наполнения и вывода) можно было проделать и в разных циклах. Например:

for(int i = 0; i <  ar1.length; i++) {  
  ar1[i] =  Math.floor(Math.random() \* 9);  
}  
for(int  i = 0; i < ar1.length; i++) {  
  System.out.print(ar1[i] + "  ");  
}

В данном случае более рационален первый способ (один проход по массиву вместо двух), но не всегда возможно выполнить требуемые действия в одном цикле.

**Самостоятельно:**

1. Изучить двумерные массивы (матрицы) (обявление и инициализацию).
2. Написать программу:

Создайте массив из 8 случайных целых чисел из отрезка [1;10]. Выведите массив на экран в строку. Замените каждый элемент с нечётным индексом на ноль. Снова выведете массив на экран на отдельной строке.