Четвертая лекция

Основные понятия объектно-ориентированного программирования

ООП позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм или наследование.

Инкапсуляция;
Наследование;
Полиморфизм.

Наследование позволяет строить новые классы на базе существующих, добавляя в них новые возможности или переопределяя существующие.

Класс, от которого производится наследование, называется базовым, родительским или суперклассом.

Новый класс — потомком, наследником, дочерним или производным классом.

Наследования можно выразить словом ЯВЛЯЕТСЯ.

В коде это пишется как **extends**.

public class Cat extends Animal { … }

Класс **Cat** наследует все поля и методы класса **Animal** (за исключением конструкторов). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.
// кроме **полей** и методов **private**.

Кроме того, в **Cat** можно:
- добавить новые поля;
- добавить новые методы;
- переопределить какие-либо методы класса **Animal**.

В Java нет множественного наследования. Т.е. у класса может быть только один базовый (супер) класс.

Подкласс в свою очередь может быть суперклассом другого подкласса.

Соответственно отношение наследования формирует строгую иерархию классов - иерархию наследования (дерево классов).

Конструкторы и наследование

Конструктор - это специальный метод, который вызывается при создании нового объекта. Конструктор нужен для автоматической инициализации переменных.

Конструкторы - не полноценные объектно-ориентированные члены, и они не наследуются; вместо этого их нужно явно реализовывать в подклассах.

Конструктор всегда имеет то же имя, что и класс, в котором он используется, и у него нет возвращаемого типа. Например:

```
public class Cat extends Animal{

    private String name;

    public Cat(String name){

        this.name = name
    }

}
```

Перегрузка конструкторов

Наряду с перегрузкой обычных методов можно также выполнять перегрузку методов конструкторов, для этого удобно один конструктор вызывать из другого через ключевое слово **this**.

```
class Rectangle{
    int height;
    int width;

    Rectangle(int h, int w) {
       height = h;
        width = w;
    }

       Rectangle(int h) {
        this(h, h);
    }

      Rectangle() {
        this(0);
    }
}
```

Использование ключевого слова super для вызова конструкторов суперкласса

В Java существует ключевое слово **super**, которое обозначает суперкласс, т.е. класс, производным от которого является текущий класс.

Подкласс может вызывать конструктор, определенный его суперклассом, с помощью следующей формы ключевого слова **super**:

super (список_аргументов);

Программа выполнит тот конструктор, который соответствует указанным аргументам.

При помощи ключевого слова super можно обратится к объекту суперкласса, скрытому объектом подкласса. (переопределенном в классе наследнике).

super.объект

Здесь объект может быть методом либо переменной экземпляра. Не стоит путать с объектами суперкласса, которые объявлены как **private**.

Чтобы исправить эту ситуацию можно объявить поля в родительском классе как **protected** и тогда мы сможем к нему обращаться из классов наследников.

Абстрактный класс

Абстрактным называется класс, на основе которого не могут создаваться объекты. При этом наследники класса могут быть не абстрактными, на их основе объекты создавать, соответсвенно, можно. Для того, чтобы превратить класс в абстрактный перед его именем надо указать ключевое слово **abstract**.

Можно создавать класс с ключевым словом **abstract** даже, если в нем не имеется ни одного абстрактного метода.

Класс содержащий хотя бы один абстрактный метод, обязан быть абстрактным.

```
public abstract class Animal {

public void voice();

}
```

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придется переопределить все абстрактные методы базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract.**

Object – базовый класс в Java.

Object – базовый класс в Java. От него наследуются все остальные классы

```
public class SimpleClass { … }
public class SimpleClass extends Object { … }
```

Object clone() - создаёт новый объект, не отличающий от клонируемого.
boolean equals(Object object) - определяет, равен ли один объект другому.
void finalize() - вызывается перед удалением неиспользуемого объекта.
getClass() - получает класс объекта во время выполнения.
int hashCode() - возвращает хэш-код, связанный с вызывающим объектом.
void notify() - возобновляет выполнение потока, который ожидает вызывающего объекта
void notifyAll() - возобновляет выполнение всех потоков, которые ожидают вызывающего объекта
void wait() - ожидает другого потока выполнения
void wait(long ms) - ожидает другого потока выполнения
void wait(long ms, int nano) - ожидает другого потока выполнения
String toString() - возвращает строку, описывающий объект

Переопределение и перегрузка (override и overload)

Ранее мы посмотрели пример того, как можно добавлять функциональность к классу путем его расширения (наследования). В класс можно добавлять новые возможности не только путем создания новых методов, а методы класса можно также переопределять — сделать **override**. Или перегрузить — сделать **overload**.

Переопределение (override)
Переопределение используется тогда, когда вы переписываете (переделываете) УЖЕ существующий метод.

Перегрузка (overload)
Перегрузка метода заключается в следующем — вы создаете метод с таким же именем, но с другим набором параметров. *Например, в классе может быть несколько методов с названием summa, но с разным набором параметров*

Массивы в Java

Массив (**англ. Array**) это объект, хранящий в себе фиксированное количество значений одного типа. Другими словами, массив — это нумерованный набор переменных. Переменная в массиве называется элементом массива, а ее позиция в массиве задается индексом.

Объявление и инициализация массива в Java

При создании массива в Java первым делом его нужно объявить. Это можно сделать следующим образом:

int[] myFirstArray;

Можно также объявить массив так:

int mySecondArray[];

Имя массива может быль любым, однако, оно должно соответствовать правилам именования в Java

Массивы можно создавать не только из переменных базовых типов, но и из произвольных объектов. например объявим массив **Cat[] cats;**

После объявления массива можно создать массив следующей сомандой.

```
myFirstArray = new int[15];
cats = new Cat[100];
```

При создании массива с помощью ключевого слова **new**, все элементы массива автоматически инициализированы нулевыми значениями.

Инициализация массива

Для того, чтобы присвоить элементам массива свои начальные значения, необходимо провести его инициализацию.

```
myFirstArray[0] = 10; //инициализация первого элемента
myFirstArray[1] = 20; // инициализация второго элемента
MyFirstArray[2] = 30; // и т. д.
```

Пример инициализации с помощью цикла.

```
for(int i = 0; i < 15; i++){
    myFirstArray[i] = 10;
}
```

Для создания и инициализации массива можно также использовать упрощенную запись. Она не содержит слово new, а в скобках перечисляются начальные значения массива.

```
int[] myColor = {255, 255, 0};
```

Определение размера массива

Размер массива не всегда очевиден, поэтому для того, чтобы его узнать следует использовать свойство **length**, которое возвращает длину массива.
Пример: Задано 4 числа, необходимо найти минимальное

```
int[] numbers = {-9, 6, 0, -59};
int min = numbers[0];
for(int i = 0; i < numbers.length; i++){
    if    (numbers[i] < min)
        min = numbers[i];
}
System.out.println(min);
```

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 в существующие элементы присвоить новые значения.

Пример реализации сортировки массива.

```
for (int i = 0; i < arr.length; i++) {
    int min = arr[i];
    int min_i = i;
for (int j = i+1; j < arr.length; j++) {
   if (arr[j] < min) {
         min = arr[j];
         min_i = j;
     }
}
if (i != min_i) {
     int tmp = arr[i];
     arr[i] = arr[min_i];
     arr[min_i] = tmp;
}
```

Сортировка массива

Существует готовое решение для сортировки массива, это метод **sort()** из класса **Arrays**, который эффективен в большинстве случаев. Для того чтобы отсортировать массив, необходимо написать всего одну строку.

Arrays.sort(arr); // где arr это имя массива

Примечание: в начале файла предварительно нужно подключить библиотеку java.util.

import java.util.*;

Класс Arrays так же содержит в себе методы с помощью которых можно манипулировать содержимым массива (поиск, заполнение, сравнение, преобразования в коллекцию и т.д.)

public static void fill(int[] a, int val) – Заполнение массива указанным элементом;

public static void sort(int[] a) – Сортировка массива;

public static boolean equals(int[] a, int[] a2) – Сравнение массивов, возвращает true если массивы равны;

public static String toString(int[] a) – Преобразовываем массив в строку;

public static int binarySearch(int[] a, int key) – Возвращает первый индекс вхождения заданного числа в отсортированном одномерном массив;

public static int[] copyOf(int[] original, int newLength) – копирование массива в новый массив с указанием какую длину массива нужно скопировать;

public static int hashCode(int[] a) – Вычисляет хэш-код исходя из значений элементов.

Матрицы и двумерные массивы в Java

Матрица это прямоугольная таблица, состоящая из строк и столбцов на пересечении которых находятся её элементы. Количество строк и столбцов матрицы задают ее размер.

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

$$A=\begin{pmatrix} 1 & -2 & 3 \\ 4 & 1 & 7 \end{pmatrix},\quad B=\begin{pmatrix} -9 & 1 & 0 \\ 4 & 1 & 1 \\ -2 & 2 & -1 \end{pmatrix}.$$

```
int[][] matrixA; matrixA = new
Int[2][3];
Пример инициализации Матрици А
matrixA[0][0] = 1;
matrixA[0][1] = -2;
matrixA[0][2] = 3;
matrixA[1][0] = 4;
matrixA[1][1] = 1;
matrixA[1][2] = 7;
```

```
int[][] matrixB = {    {-9,1,0},
{4,1,1},
{-2,2,-1}
};
```

Для того, чтобы вывести матрицу на консоль, нужно пройти все элементы, используя два цикла. Количество циклов, при прохождении элементов массива, равно его размерности. В нашем случае первый цикл осуществляется по строкам, второй — по столбцам.

```
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        System.out.print(matrixA[i][j] + "\t");
    }
    System.out.println();
}
```

"\t" — табуляция

Длину двухмерного массива можно узнать **matrixA**[0].length;