Лабораторная работа №5
Классы – оболочки. Массивы. Передача параметров методам
Классы - оболочки примитивных типов данных

Java использует встроенные примитивные типы данных, например, int или char, ради обеспечения высокой производительности. Эти типы данных не принадлежат к классовой иерархии Java. Для каждого примитивного типа в Java реализован специальный класс с таким же или похожим названием, которое пишется с прописной буквы (Integer, Character, Boolean и т.п.). Эти классы, которые часто называют оболочками или обертками, имеют дополнительные методы обработки типов данных. С классами-оболочками нельзя выполнять арифметические действия. Нужно сначала получить доступ к значениям базовых типов, заключенных в класс-оболочку.

Абстрактный класс **Number** представляет собой интерфейс для работы со всеми стандартными скалярными типами: — long, int, float и double.
• doubleValue() возвращает содержимое объекта в виде значения типа double.
• floatValue() возвращает значение типа float.
• intValue() возвращает значение типа int.
• longValue() возвращает значение типа long.

Double и Float

Double и Float — подклассы класса Number. В дополнение к четырем методам доступа, объявленным в суперклассе, эти классы содержат несколько сервисных методов, которые облегчают работу со значениями double и float. У каждого из классов есть конструкторы, позволяющие инициализировать объекты значениями типов double и float, кроме того, для удобства пользователя, эти объекты можно инициализировать и объектом String, содержащим текстовое представление вещественного числа. Приведенный ниже пример иллюстрирует создание представителей класса Double с помощью обоих конструкторов.

```
class DoubleDemo {
public static void main(String args[]) {
Double d1 = new Double(3.14159);
Double d2 = new Double("314159E-5");
System.out.println(d1 + " = " + d2 + " -> " + d1.equals(d2));//equals применим для сравнения объектов
} }
```

Как вы можете видеть из результата работы этой программы, метод equals возвращает значение true, а это означает, что оба использованных в примере конструктора создают идентичные объекты класса Double.

```
3.14159 = 3.14159 -> true
```

Бесконечность и NaN

В спецификации IEEE для чисел с плавающей точкой есть два значения типа double, которые трактуются специальным образом: бесконечность и NaN (Not a Number — неопределенность). В классе Double есть тесты для проверки обоих этих условий, причем в двух формах — в виде методов (статических), которым значение double передается в качестве параметра, и в виде методов, проверяющих число, хранящееся в объекте класса Double.
• isInfinite(d) возвращает true, если абсолютное значение указанного числа типа double бесконечно велико.
• isInfinite() возвращает true, если абсолютное значение числа, хранящегося в данном объекте Double, бесконечно велико.
• isNaN(d) возвращает true, если значение указанного числа типа double неопределено.
• isNaN() возвращает true, если значение числа, хранящегося в данном объекте Double, неопределено.

Очередной наш пример создает два объекта Double, один с бесконечным, другой с неопределенным значением.

```
class InfNaN {
public static void main(String args[]) {
Double d1 = new Double(1/0.);
Double d2 = new Double(0/0.);
System.out.println(d1 + ": " + d1.isInfinite() + ", " + d1.isNaN());
System.out.println(d2 + ": " + d2.isInfinite() + ", " + d2.isNaN());
} }
```

Ниже приведен результат работы этой программы:

```
Infinity: true, false
NaN: false, true
```

Integer и Long

Класс Integer — класс-оболочка для чисел типов int, short и byte, а класс Long — соответственно для типа long. Помимо наследуемых методов своего суперкласса Number, классы Integer и Long содержат методы для преобразования текстового представления чисел, и наоборот, для представления чисел в виде текстовых строк. Различные варианты этих методов позволяют указывать основание (систему счисления), используемую при преобразовании. Обычно используются двоичная, восьмеричная, десятичная и шестнадцатиричная системы счисления.

- parseInt(String) преобразует текстовое представление целого числа, содержащееся в переменной String, в значение типа int. Если строка не содержит представления целого числа, записанного в допустимом формате, вы получите исключение NumberFormatException.
- parseInt(String, radix) выполняет ту же работу, что и предыдущий метод, но в отличие от него с помощью второго параметра вы можете указывать основание, отличное от 10.
- toString(int) преобразует переданное в качестве параметра целое число в текстовое представление в десятичной системе.
- toString(int, radix) преобразует переданное в качестве первого параметра целое число в текстовое представление в задаваемой вторым параметром системе счисления.

Character

Character — класс-оболочка типа char. У него есть несколько полезных статических методов, с помощью которых можно выполнять над символом различные проверки и преобразования.

- isLowerCase(char ch) возвращает true, если символ-параметр принадлежит нижнему регистру.
- isUpperCase(char ch) делает то же самое в случае символов верхнего регистра.
- isDigit(char ch) и isSpace(char ch) возвращают true для цифр и пробелов, соответственно.
- toLowerCase(char ch) и toupperCase(char ch) выполняют преобразования символов из верхнего в нижний регистр и обратно.

Ниже показаны примеры использования некоторых методов классов-оболочек типов:

```
public static void main(String[] args) {
            String s= "10";

            int a=Integer.parseInt(s);
            int b=Integer.parseInt(s,16);

            System.out.println("s перед вызовом parseInt(s: " + s) ;
            System.out.println("a после вызова parseInt(s): " + a);
            System.out.println("b после вызова parseInt(s,16): " + b);

            String s1, s2;
            s1=Integer.toString(a);
            s2=Integer.toString(b, 16);

            System.out.println("a после вызова toString: " + s1);
            System.out.println("b после вызова toString: " + s2);

            Double dt = new Double(9.87);
            Integer it=new Integer(55);
```

```
        double d=dt.doubleValue();//возвращает значение типа double.
        int i=it.intValue(); //возвращает значение типа int.

         System.out.println("dt после вызова doubleValue(): " + d);
         System.out.println("it после вызова intValue(): " + i);

        i=Integer.valueOf("235").intValue();
              // intValue()возвращает значение типа int.
              //valueOf возвращает значение типа Integer
        System.out.println("i после вызова valueOf().intValue(): " + i);
    }
```

Массивы

Массивы (arrays) представляют группы однотипных переменных. Массив это сложный тип, который может состоять из нулевого или ненулевого количества элементов другого типа. Если этот другой тип тоже представляет собой массив, то первый массив называется *многомерный*.
Элементы массива могут принадлежать к примитивному типу, например float, char или int. Кроме того, элементы могут иметь тип массива, класса или интерфейса.
Переменные входящие в массив, называют элементами массива. На элемент массива можно сослаться, указав его место в массиве (по номеру элемента массива) как, например, во фразе “третий пункт списка покупок”.
Номер элемента массива называется индексом (индексы массива начинаются с нуля и увеличиваются до значения, которое на единицу меньше длины массива).
Синтаксис объявления массива:

```
Тип имя_массива[];
```

Тип – тип элементов, составляющих массив. Тип и число элементов массива определяют объем памяти необходимый для размещения массива.
В качестве имени_массива может выступать любой идентификатор.
При объявлении массива пара квадратных скобок, определяющих массив может располагаться как после имени массива так и перед его именем. Например, следующие объявления идентичны:

```
int d[]; //Объявление массива целых чисел
int []d; //Объявление массива целых чисел
```

Здесь объявлен массив без выделения области памяти, для размещения его элементов. Данное объявление означает только то, что переменная d будет служить ссылкой на массив, состоящий из целых чисел.
В следующем примере объявляются сразу несколько массивов одного и того же типа:

```
int[] array1,array2, array3;
```

Заполнять элементы массива значениями можно также по-разному:

```
int[] b={3,2,1};
String s1=”One”;
String s2=”Two”;
String c[]={s1,s2};
```

Для того, чтобы зарезервировать память под массив, используется специальный оператор new. В приведенной ниже строке кода с помощью оператора new массиву month_days выделяется память для хранения двенадцати целых чисел.

```
month_days = new int [12];
```

Итак, теперь month_days — это ссылка на двенадцать целых чисел. Ниже приведен пример, в котором создается массив, элементы которого содержат число дней в месяцах года (невисокосного).

```
class Array {
public static void main (String args []) {
int month_days[];
month_days = new int[12];
month_days[0] = 31;
month_days[1] = 28;
month_days[2] = 31;
month_days[3] = 30;
month_days[4] = 31;
month_days[5] = 30;
month_days[6] = 31;
month_days[7] = 31;
month_days[8] = 30;
month_days[9] = 31;
month_days[10] = 30;
month_days[11] = 31;
System.out.println("April has " + month_days[3] + " days.");
} }
```

При запуске эта программа печатает количество дней в апреле, как это показано ниже. Нумерация элементов массива в Java начинается с нуля, так что число дней в апреле — это month_days [3].

```
April has 30 days.
```

В результате работы этой программы, вы получите точно такой же результат, как и от ее более длинной предшественницы.

```
class AutoArray {
public static void main(String args[]) {
int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
System.out.println("April has " + month_days[3] + " days.");
} }
```

Java строго следит за тем, чтобы вы случайно не записали или не попытались получить значения, выйдя за границы массива. Если же вы попытаетесь использовать в качестве индексов отрицательные числа либо числа, которые больше или равны количеству элементов в массиве, то получите сообщение об ошибке времени выполнения.

Многомерные массивы

На самом деле, настоящих многомерных массивов в Java не существует. Зато имеются массивы массивов, которые ведут себя подобно многомерным массивам. Приведенный ниже код создает традиционную матрицу из шестнадцати элементов типа double, каждый из которых инициализируется нулем. Внутренняя реализация этой матрицы — массив массивов double.

```
double matrix [][] = new double [4][4];
```
Следующий фрагмент кода инициализирует такое же количество памяти, но память под вторую размерность отводится вручную. Это сделано для того, чтобы наглядно показать, что матрица на самом деле представляет собой вложенные массивы.

```
double matrix [][] = new double [4][];
            matrix [0] = new double[4];
            matrix[1] = new double[4];
            matrix[2] = new double[4];
            matrix[3] = new double[4];
```

В следующем примере создается матрица размером 4 на 4 с элементами типа double, причем ее диагональные элементы (те, для которых x==y) заполняются единицами, а все остальные элементы остаются равными нулю.

```
class Matrix {
public static void main(String args[]) {
double m[][];
m = new double[4][4];
m[0][0] = 1;
m[1][1] = 1;
m[2][2] = 1;
m[3][3] = 1;
System.out.println(m[0][0] +" "+ m[0][1] +" "+ m[0][2] +" "+ m[0][3]);
System.out.println(m[1][0] +" "+ m[1][1] +" "+ m[1][2] +" "+ m[1][3]);
System.out.println(m[2][0] +" "+ m[2][1] +" "+ m[2][2] +" "+ m[2][3]);
System.out.println(m[3][0] +" "+ m[3][1] +" "+ m[3][2] +" "+ m[3][3]);
}
}
```
Запустив эту программу, вы получите следующий результат:

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```
Обратите внимание — если вы хотите, чтобы значение элемента было нулевым, вам не нужно его инициализировать, это делается автоматически.
Для задания начальных значений массивов существует специальная форма инициализатора, пригодная и в многомерном случае. В программе, приведенной ниже, создается матрица, каждый элемент которой содержит произведение номера строки на номер столбца. Обратите внимание на тот факт, что внутри инициализатора массива можно использовать не только литералы, но и выражения.

```
class AutoMatrix {
public static void main(String args[]) {
                        double m[][] = {
                        { 0*0, 1*0, 2*0, 3*0 }, { 0*1, 1*1, 2*1, 3*1 }, { 0*2, 1*2, 2*2, 3*2 },
                        { 0*3, 1*3, 2*3, 3*3 } };
                        System.out.println(m[0][0] +" "+ m[0][1] +" "+ m[0][2] +" "+ m[0][3]);
                        System.out.println(m[1][0] +" "+m[1][1] +" "+ m[1][2] +" "+ m[1][3]);
                        System.out.println(m[2][0] +" "+m[2][1] +" "+ m[2][2] +" "+ m[2][3]);
                        System.out.println(m[3][0] +" "+m[3][1] +" "+ m[3][2] +" "+ m[3][3]);} }
```
Запустив эту программу, вы получите следующий результат:

```
0 0 0 0
0 1 2 3
0 2 4 6
0 3 6 9
```

Передача параметров методам

В общем случае, существует два способа, которыми компьютерный язык может передавать аргументы подпрограмме – по значении и по ссылке. В Java элементарный (примитивный) тип передается методу по значению. К ссылочным типам данных относятся *массивы, классы и интерфейсы* .

Первый способ — вызов по значению. При использовании этого подхода значение аргумента копируется в формальный параметр подпрограммы. Следовательно, изменения, выполненные в параметре подпрограммы, не влияют на аргумент. Второй способ передачи аргумента — вызов по ссылке. При использовании этого подхода параметру передается ссылка на аргумент (а не его значение). Внутри подпрограммы эта ссылка используется для обращения к реальному аргументу, указанному в вызове. Это означает, что изменения, выполненные в параметре, будут влиять на аргумент, использованный в вызове подпрограммы. Как вы убедитесь, в Java применяются оба подхода, в зависимости от передаваемых данных.
В Java элементарный (примитивный) тип передается методу по значению. Таким образом, все происходящее с параметром, который принимает аргумент, не оказывает влияния вне метода. Например, рассмотрим следующую программу:

```
// Значения примитивных типов передаются по значению.
class Test {
void meth(int i, int j) {
i *= 2;
j /= 2;
}
}
class CallByValue {
public static void main(String args[]) {
Test ob = new TestO ;
int a = 15, b = 20;
System.out.println("a и b перед вызовом: " + a + " " + b) ;
ob.meth(a, b);
System.out.println ("a и b после вызова: " + a + " " + b);
}
}
```
Вывод этой программы имеет следующий вид:

```
a и b перед вызовом: 15 20
a и b после вызова: 15 20
```

Выполняемые внутри метода meth () операции не влияют на значения a и b, использованные в вызове. Их значения не изменились на 30 и 10.
При передаче объекта методу ситуация изменяется, поскольку, по существу, объекты передаются посредством вызова по ссылке. Следует помнить, что при создании переменной типа класса создается лишь ссылка на объект. Таким образом, при передаче этой ссылки методу, принимающий ее параметр будет ссылаться на тот же объект, на который ссылается аргумент. Изменения объекта внутри метода влияют на объект, использованный в качестве аргумента. Например, рассмотрим такую программу:

```
// Объекты передаются по ссылке
class Test
{ int a, b;
Test(int i, int j ) {
a = i; b = j ;
}
// передача объекта
void meth(Test o) {
o.a *= 2;
o.b /= 2;
}
}
class CallByRef {
public static void main(String args[]) {
Test ob = new Test(15, 20);
System.out.println("ob.a и ob.b перед вызовом: " + ob.a + " " + ob.b) ;
ob.meth(ob);
System.out.println("ob.a и ob.b после вызова: " + ob.a +" "+ ob.b);
}
}
```

Эта программа генерирует следующий вывод:

```
ob.a и ob.b перед вызовом: 15 20
ob.a и ob.b после вызова: 30 10
```

В данном случае действия внутри метода meth () влияют на объект, использованный в качестве аргумента.
Все значения объектов оболочек-типов, как и значения примитивных типов данных, в java передаются по значению. Это значит что метод, получает копии всех параметров и работает только с ними.

```
public static void main(String[] args) {
Integer d=new Integer(5);
System.out.println("d перед вызовом: " + d) ;
 change(d);
 System.out.println("d после вызова: " + d);
}
public static void change(Integer x) {
 x = 6;
 }
```

Эта программа генерирует следующий вывод:

```
d перед вызовом: 5
d после вызова: 5
```

Тоже относится и к объектам типа String:

```
public static void main(String[] args) {
String s = "aaa";;
System.out.println("s перед вызовом: " + s) ;
change(s);
System.out.println("s после вызова: " + s);
}
public static void change(String x) {
   x = "bbb";
  }
```

Эта программа генерирует следующий вывод:

```
s перед вызовом: aaa
s после вызова: aaa
```

В этом случае передается значение ссылки на s. Создается копия значения ссылки - переменная x. Так как и s и x ссылаются в одно место, значение строки должно меняться в методе change(). Но оно не меняется, потому что объекты класса String - immutable, то есть неизменные. Это значит, что если мы создадим объект класса String, то значение этого объекта мы уже поменять не сможем. Сможем лишь создать новый объект класса String, на который будет ссылаться ссылка. Это же относится к Integer, Long и ко всем оболочкам примитивных типов.

В Java параметры- массивы передаются методам по ссылке. В этом примере меняется значение первого элемента массива:

```
public static void change(int[] x) {
x[0] = 7;
}

public static void main(String[] args) {
int a[] = new int[1];
a[0] = 5;
System.out.println(a[0]);
change(a);
System.out.println(a[0]);
}
```

Ниже показан пример интерфейса, который передается в качестве параметра. Если интерфейс в параметре, то мы можем туда передать любой экземпляр класса, имплементирующий указанный интерфейс.

```
interface Berry{
   public int getSugar();
}

class Strawberries implements Berry {
   public int getSugar() {
```

```
        return 20;
    }
}

class Watermelon implements Berry {
    public int getSugar() {
        return 50;
    }
}

public class abstr{
            public static void main(String[] args) {
                        abstr abc =new abstr();
                        abc.test();
            }
    public void test(){
        printBerrySugar(new Strawberries ());
        printBerrySugar(new Watermelon ());
    }

    public void printBerrySugar(Berry berry){
        System.out.println(berry.getSugar());
    }
}
```

Задание 1.

В классе вызвать три метода: первый получает параметр – целое число (примитивного типа данных), второй – объект (должен быть вами создан), с которым передается переменная объекта - целое число и третий – второй элемент массива целых чисел. В каждом из методов значение параметра меняется: в первом – множится на два, во втором – на три и в третьем – на четыре. Вывести на экран значения всех трех параметров до и после того, как указанные методы закончат свою работу. Проанализировать результаты передачи параметров по ссылке и по значению.