Лабораторная работа N 2

Синтаксис и структуры данных языка Java

1. Введение

Данная работа посвящена описанию лексики языка Java. Лексика описывает, из чего состоит текст программы, каким образом он записывается и на какие простейшие слова (лексемы) компилятор разбивает программу при анализе. Лексемы – это основные "кирпичики", из которых строится любая программа на языке Java. В лабораторной работе раскрываются многие детали внутреннего устройства языка.

Целью лабораторной работы является написание программы на языке Java с применением различных конструкций языка выполняющую обработку данных.

2. Общие сведения

2.1 Синтаксис языка Java. Лексемы.

Во время компиляции компилятор Java анализирует текст программы и извлекает для последующей обработки определенные блоки информации, называемые лексемами. Прежде всего, из исходного потока байтов выделяются `escape`-последовательности и проверяется, обозначают ли они конец строки или являются входными символами. Затем он устраняет пробелы, не входящие в состав строковых литералов, включая символы пробела, возврата каретки, перевода строки и горизонтальной табуляции. После этого `javac` приступает к извлечению лексем.

Существует несколько типов лексем: *`литералы, идентификаторы, ключевые слова, символы операций, разделители и комментарии.`*

Большинство программистов при работе с Java используют следующее неформальное правило: имена классов записывать с прописной буквы, а имена переменных и функций со строчной буквы. Прописные буквы или символы подчеркивания используют для разделения слов в многословных именах, например, `secretPassword` или `some_thing.`

Ключевые слова (keywords) – являются зарезервированными. Это означает, что их нельзя использовать никаким другим способом, помимо того, для которого они предназначены спецификацией Java (поскольку эти слова зарезервированы их нельзя употреблять в качестве идентификаторов).

Зарезервированные ключевые слова Java

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

Объявление переменной

<спецификатор доступа> <тип> [final] <имя>[=<значение>];

Область действия:

- областью действия переменной является блок в котором она объявлена;
- вложенный блок не может иметь переменной с именем, которое уже используется во внешнем блоке;

2.2. Литералы.

Программисты часто используют в качестве величины одиночный символ. В Java такие величины представляются *символьными литералами (character literals).* Значение символьного литерала заключается в одиночные кавычки, например `'y'`.

Задание символьного литерала становится более интересным, если его значением должен быть символ одиночной кавычки, обратный слэш или другой непечатаемый символ. Символ обратного слэша (\) позволяет определить некоторые непечатаемые символы или символы, которые обычно являются частью команды. Например, значение '\'' обозначает одиночную кавычку. В представлены некоторые примеры задания значений символьных литералов.

Задание символьных литералов:

Описание escape-последовательности	Последовательность	Результат
Любой символ	' y '	y
Возврат (BS)	'\b'	Возврат (backspace)
Горизонтальная табуляция (HT)	'\t'	Табуляция (tab)
Перевод строки (LF)	'\n'	Перевод строки (linefeed)
Перевод страницы (FF)	'\f'	Перевод страницы (from feed)
Возврат каретки (CH)	'\r'	Возврат каретки (carriage return)
Двойная кавычка	'\''	"
Одиночная кавычка	'\''	'
Обратный слэш	'\\'	\
Восьмеричный набор битов	'\ddd'	Символ с восьмеричным значением ddd
Шестнадцатеричный набор битов	'\xdd'	Символ с шестнадцатеричным значением dd
Символ Unicode	'\udddd'	Символ с кодом dddd в кодировке Unicode

Строковые литералы (string literals) представляют собой последовательности символов, заключенные в двойные кавычки, например `"Это строковый литерал"`. Другие возможные литералы — это `"Hello, World"` или даже `" "`, обозначающий

пустую символьную строку. Компилятор `javac` не удаляет пробелы внутри строковых литералов.

Для строковых литералов определена операция **конкатенации** (или сцепления). Например, пусть одна строка содержит символы `"This is the beginning"`, а другая - символы `" of a beautiful relationship"`. Эти строки можно сцепить вместе с помощью записи `"This is the beginning" + " of a beautiful relationship".` (Пробелы по обеим сторонам знака плюс необязательны.)

Строковый литерал не может располагаться на нескольких строках. Если это необходимо, его можно при объявлении разбить на несколько литералов, а затем сцепить при использовании. Это также облегчает чтение программы и снижает риск принять по ошибке строковые литералы за комментарии.

2.3. Операции

Символы операций (operators) – это символы, которые применяются для обозначения арифметических и логических операций. Арифметические символы определяют операции применяемые к числам (например, 2+3). Символы операций, за исключением знака плюс (+), используются только для арифметических вычислений. Операция + может также использоваться для сцепления строк.

Основные арифметические операторы

Оператор	Краткое описание	Пример
+	Сложение	3+2 или “pre”+”fix”
-	Вычитание	12-3
*	Умножение	length*width
/	Деление	miles/gallons
%	Оператор взятия модуля, определяет остаток от деления первого целочисленного операнда на второй	10%4
&	Побитовое И	num&musk
|	Побитовое ИЛИ	this|that
^	Побитовое исключающее ИЛИ	tall^short
~	Побитовое дополнение	mask=~item
<<	Сдвиг влево	var<<3
>>	Сдвиг вправо	var>>12
<<<	Сдвиг влево с добавлением нулей	arg<<<howMuch
>>>	Сдвиг вправо с добавлением нулей	Arg>>>howMuch

Булевы операции

Символ	Действие	Пример
==	равно	if (a==14)
!=	не равно	if (a!=14)
<	меньше чем	if (a<14)
>	больше чем	if (a>14)
<=	меньше или равно	if (a<=1)
>=	больше или равно	if (a>=4)
&	побитовое и	if ((a>14)&(a<17))
&&	и	if((a>14)&&(b>17))
|	побитовое или	if((a==16)|(b==19))
||	или	if ((a==25)||(a==8))
!	нет	if (!((a==16)||(a==3)))
^	исключающее или	if ((a==16)^(a==19))

Оператор присваивания

Оператор	Эквивалентный вид оператора
a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a%=b	a=a%b
a&=b	a=a&b
a|=b	a=a|b
a^=b	a=a^b
a<<=b	a=a<<b
a>>=b	a=a>>b

Приоритет и ассоциативность операций

Высший приоритет					
оператор	**ассоциативность**				
. ( ) [ ]	слева направо	<< >> <<< >>>	Слева направо	|	слева направо
++ -- ! ~	слева направо	< > <= >=	Слева направо	&&	слева направо
new		== !=	Слева направо	||	слева направо
* / %	слева направо	&	Слева направо	? :	слева направо
+ -	слева направо	^	Слева направо	= += -= *= /= %= &= ||= <<= >> = ^=	Справа налево
				низший приоритет	

2.4 Типы данных

Java является сильнотипизированным языком. *Сильнотипизированными* языками, называются языки требующие, чтобы программист, перед тем как использовать какие-либо переменные определил их тип. Некоторые языки, например BASIC, являются слаботипизированными. В таких языках компьютер определяет, к какому типу относятся данные, уже непосредственно в процессе обработки.

2.4.1 Простые типы данных

В таблице приведены простые типы данных языка Java.

Ключевое слово	Тип	Размер	Значения, которые может хранить	Значение по умолчанию
byte	байт	8 бит	значения в диапазоне от -2^7 до 2^7-1	0
short	короткое целое	16 бит	значения в диапазоне от -2^{16} до $2^{16}-1$	0
int	целое	32 бит	значения в диапазоне от -2^{32} до $2^{32}-1$	0
long	длинное целое	64 бит	значения в диапазоне от -2^{64} до $2^{64}-1$	0
float	плавающая точка	32 бит	значения в диапазоне от$1.7*10^{-38}$ до $1.7*10^{38}$	0.0f
double	удвоенная точность	64 бит	значения в диапазоне от *-1.40239846E-45 до 3.40282347E+38*	0.0d
char	Символьный	16 бит	буквы, цифры, символы, основанные на 16-битном наборе символов Unicode от /u0000 до /uffff	'0x0'
boolean	логический (булевой)	8 бит	true либо false	false

Логический (булевой) литерал (boolean literal) представляет собой одно из двух слов: true (истина) или false (ложь). В отличие от других языков программирования (например

Pascal), эти слова не связаны с какими-либо числовыми значениями, например 0 и 1. Таким образом, значение логического литерала буквально представляет собой истину или ложь.

Символьный тип данных. Тип `char` в действительности является 16-разрядным беззнаковым целым, которое представляет символ в кодировке **unicode**. Другими словами, символы unicode и escape-коды можно задавать с помощью цифрового представления. Помните, что значение unicode определено для всех печатаемых и непечатаемых символов. Благодаря тому, что Java хранит все символы в формате Unicode, его можно использовать практически с любым существующим в мире письменным языком и это одна из сильных сторон Java.

Вышеперечисленные типы данных относятся к *простым типам данных* Java. Если программист явно не указал начальное значение переменной , Java инициализирует все примитивные типы данных значениями по умолчанию. Целочисленные переменные и переменные с плавающей точкой инициализируются значением 0. Тип char получает значение `null`, а тип `boolean` – значение `false`.

2.4.2 Ссылочные типы данных

Кроме этих типов в Java существуют еще так называемые ***ссылочные типы данных(reference data types).*** К ссылочным типам данных относятся *массивы, классы и интерфейсы* **.**

Ссылочные типы данных определяют *адрес*, который указывает на набор данных. Эти данные в свою очередь могут иметь примитивный или ссылочный тип.

Объявление переменной ссылочного типа в Java не выделяет автоматически память под переменную.

2.4.3 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.

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. Второй вариант больше гармонирует со способом определения объектов в языке Java. Например, следующие объявления идентичны:

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

Мы объявили массив, не выделяя область 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;
```

Прежде чем использовать массив, необходимо в процессе выполнения программы отвести под него память с помощью оператора `new`, который создает в памяти новый экземпляр ссылочного типа данных.

Например:

```
d=new int[10];
```

Этот оператор создает массив из десяти целых чисел и присваивает переменной `d` значение адреса первого элемента массива. Каждый элемент массива инициализируется по правилам умолчания, предусмотренным для каждого типа данных. В этом случае все переменные становятся равными нулю. (Массив объектов какого-то класса по умолчанию инициализируется значениями `null`).

Пример одновременного объявления и создания массива с помощью оператора `new`:

```
Foo[] c=new Foo[10]; // массив указателей на объекты
String b[]=new String[10];
```

Для обращения к отдельному элементу массива используется индекс, заключенный в квадратные скобки. Например, с помощью оператора присваивания, таким образом, можно присвоить значения элементам массива d:

```
d[0]=1;
d[1]=2;
d[2]=6;
d[3]=5;
d[4]=1;
...
d[9]=7;
```

Заполнять элементы массива значениями можно также таким образом:

```
int [] a;
a={1,3,67, 77,11,2};
int [] b={3,2,1};
String s1="One";
String s2="Two";
String c[]={s1,s2};
```

Массив `a` заполняется пятью целыми числами, а массив `b` – тремя. В следующих двух строках программы определяются три строчные переменные, значения которых в четвертой строке присваивается элементам массива `c`.

Следующий фрагмент кода демонстрирует объявление массива, его создание и присвоение значений элементам массива:

```
class Array{
    public static void main (String args[]){
        int LISTSIZE =5;
        String[] ShoppingList;
        int i=LISTSIZE;
        // создание массива
        ShoppingList = new String[LISTSIZE];
        //Инициализация массива
        ShoppingList[0] = "carrots";
        ShoppingList[1] = "tofu";
        ShoppingList[2] = "rise milk";
        ShoppingList[3] = "onions";
        ShoppingList[4] = "pasta noodles";
        for (i=0; i<LISTSIZE;i++)
        System.out.println(ShoppingList[i]);
    }
}
```

Размер массива может быть переменным:

```
int n;
// … переменной n присваивается положительное значение.
int intArray[]=new int[n];
```

В приведенном фрагменте кода объявляется и создается массив `intArray` состоящий из `n` элементов. Значение `n` вычисляется (или даже вводится пользователем) в фрагменте программы, которой здесь не приводится. Итак, массивы Java полностью динамические и их размеры вычисляются в ходе выполнения программы.

Изменить размер созданного массива нельзя. Тем не менее можно создать новый массив и присвоить ссылку на него существующей переменной.

```
intArray=new int[250]; //Создание нового массива
```

Система сборки мусора в Java автоматически находит неиспользуемые элементы и освобождает заданную ими память для других переменных. Поскольку массивы создаются во время выполнения программы, рекомендуется использовать в условных циклах и в других операторах переменную `length`, доступную для всех массивов. Следующий оператор устанавливает значения элементов массива равными их индексам:

```
for (int i=0; i<intArray.length;i++)
intArray[i]=i;
```

Переменная `intArray.length` равна количеству элементов массива.

Двумерный массив рассматривается в языке Java как массив одномерных массивов. Теоретически все массивы в Java одномерны (мы отмечаем здесь эту характеристику Java поскольку в технической документации Java отмечено, что все массивы одномерны). Можно определить массив массивов, который будет выступать как двумерный массив. Для двухмерных массивов необходимо две ссылки.

Пример, объявления двумерного массива:

```
int d[][];
```

Двумерный массив напоминает набор ячеек электронной таблицы. Чтобы сослаться на элемент, нужно указать номера строки и столбца, где он находится, например `[3][4]`.

В Java строки массива не обязаны иметь одинаковую длину, как в большинстве языков программирования. Это является следствием и одним из преимуществ динамического создания массивов во время выполнения программы. Например, можно создать *треугольный* массив.

Пример объявления и создания двумерных массивов:

```
int ticTacToeBoard[][]=new int [3][3];
int AgeWeight [][]=new int[100][]; //Фактически отводится память для ста
ссылок на массивы типа int
int AgeWeight [0][]=new int[3];
int AgeWeight [1][]=new int[5];
//…
```

В примере создается два двухмерных массива: ticTacToeBoard размерности 3 на 3 и AgeWeight, для которого определена только размерность первого индекса.

Естественно, многомерные массивы могут иметь и больше двух индексов (измерений). Например, в следующем примере объявляется пятимерный массив чисел с плавающей точкой `fifthDimention`:

```
double fifthDimention[][][][][];
```

2.5.Операторы

Во приложениях Java используются следущие основные операторы управления потоком *if (если)*, *for (для)*, *while (пока)*.

2.5.1 Оператор if

Синтаксис оператора if:

```
if (<выражение>) <оператор1>;
else <оператор2>;
/*else-часть может отсутствовать.*/
```

Сначала вычисляется `выражение`, и если оно истинно выполняется $оператор_1$. Если выражение ложно и существует `else`-часть, то выполняется $оператор_2$. В отличие от языка С в языке Java выражение в условных конструкциях должно всегда возвращать булево(логическое) значение Нельзя использовать нулевое и не нулевое значение числовой переменной как замену для true и false. В качестве оператора может выступать отдельный оператор или блок операторов. Отдельный оператор всегда должен заканчиваться точкой с запятой (ограничителем операторов). Если используется блок операторов, то он должен быть заключен в фигурные скобки.

2.5.2 Оператор switch

Оператор `switch` обеспечивает передачу управление в одну из нескольких точек программы в зависимости от значения выражения.

`Синтаксис оператора switch (переключатель):`

```
switch (выражение){
case константа_1: оператор;
break;
case константа_2: оператор;
break;
….
default: оператор;
}
```

Значение выражения сопоставляется со всеми находящимися внутри `switch`-оператора `case`-константами. Оператор, указанный после `case`-метки, выполняется, если значение `case`-выражения равно соответствующей константе. Если ни с одной из `case`-констант совпадения нет, то управление передается на конструкцию с `default`-меткой, при условии ее наличия, в противном случае ни одна из под инструкций `switch` не выполняется. Так как `case`-константы по существу являются метками, то после найденного совпадения операторы будут выполняться последовательно до тех пор, пока не закончиться `switch`-оператор, поэтому после каждого конкретного блока операторов, относящегося к конкретной `case`-метке, для того чтобы выйти за пределы оператора `switch` необходимо указать ключевое слово (оператор) `break`.

2.5.3 Оператор for

Синтаксис оператора for:

```
for(выражение_1; выражение_2; выражение3) оператор;
```

В `выражении_1` производится инициализация переменных цикла. Если `выражение_1` включает несколько операторов, то они должны быть разделены запятыми. В качестве `выражения_1` применяются присваивания или вызовы функций.

`Выражение_2` формулирует условие выполнения цикла (т.е. если `выражение_2` истинно, то операторы тела цикла выполняются, в противном случае происходит выход из цикла). Если `выражение_2` опущено, то считается, что его значение всегда истинно.

`Выражение_3` выполняется после каждого прогона цикла. Здесь обычно выполняется приращения переменных цикла.

Примеры:

```
for (int i=1; i<7; i++){
System.out.println(x);
}
```

Переменные, созданные в операторе `for` являются локальными в блоке цикла.

```
for(i=0; IsTrustIt(item[i]), i++);
```

Осуществляет просмотр элементов массива. Оператор не имеет блока кода и выполняет метод `IsTrustIt`, пока не найдет нужный элемент.

Два эти примера имеют существенные различия: в первом ключевое слово `int` инициализирует переменную `i` как локальную для оператора `for`, а во втором переменная `i` должна быть определена вне оператора `for`, и после его завершения она сохраняет свое значение.

2.5.4 Оператор while

Синтаксис оператора *while*:

```
while(выражение)оператор;
```

Если результат булевого `выражения` – `true`, то выполняется оператор тела цикла, после чего `выражение` вновь вычисляется.

Пример оператора `while`:

```
int x=100;
while (x>0){
    System.out.println(x--);
}
```

Сначала проверяется `условие` выполнения цикла (`x>0`), и если его значение `true`, то выполняется тело цикла, и процесс повторяется пока условие не станет `false`.

2.5.5 Оператор do-while

В цикле `do` сначала выполняется тело цикла, а затем – проверка условия (этот цикл называется еще цикл с предпроверкой условия). В остальном процесс выполняется аналогично работе оператора `while`.

```
do оператор;
while(выражение) оператор;
```

Оператор `do-while` будет выполнен хотя бы один раз.

Пример оператора `do-while`:

```
do{
    System.out.println(time);
    time--;
}while(time>0);
```

2.5.6 Оператор условия

Оператор условия имеет три операнда и в некоторых случаях может заменить собой блок `if-else`. Первый операнд отделяется от остальных знаком вопроса(“?”), а второй от третьего символом “двоеточие”(“:”):

```
int PosValue = x>0? x:0;
float Avg=num>1?total/num:num;
```

Выполнение оператора условия начинается с определения булевого значения первого операнда. Если оно равно `true`, то вычисляется значение второго операнда, а если false то третьего. Результат вычисления присваивается переменной. В первом примере при `x=10` `PosValue` получит значение `10`, а при `x=-1` значение 0.Во втором примере деление будет произведено только при `num>1`, что позволит избежать ненужных

вычислений в программе. Заметим, что арифметические выражения и операции сравнения имеют больший приоритет, чем оператор условия, и, следовательно, выполняются перед ветвлением программы в операторе условия.

2.5.7 Операторы break и continue

Оператор break заставляет интерпретатор Java сразу перейти к окончанию составного оператора, в который входит этот оператор break. Мы уже рассмотрели применение оператора break с оператором switch. Оператор break чаще всего записывают в виде ключевого слова break и точки с запятой:

```
break;
```

В таком виде он заставляет интерпретатор Java выйти из ближайшего цикла while, do, for или оператора switch. Например:

```
for(int i - 0; i < data.Length;  i++) {  // Цикл по массиву data
 if(data[i] == target) {                 // Если мы нашли то, что искали,
  Index = i;                            //запомним, где мы его нашли,
 break;                                // и остановим поиск!
 }
}   // После выполнения break интерпретатор Java переходит сода.
```

В языке Java нет оператора goto. Вместо него после оператора break может стоять метка составного оператора. В таком виде оператор break заставляет интерпретатор Java немедленно выйти из указанного блока, который может быть любым оператором, а не только оператором switch или циклом. Например:

```
testfornull: if (data != null) {             // Если массив определен,
 for(int row=0; row<numrows; row++) {    // цикл по строкам,
  for(int col=0; col<numcols; col++)  { // затем по столбцам.
   if(date[row][col] == null)          // Если в массиве нет данных,
    break testfornull;      // считаем, что массив не определен.
 }
} // После выполнения break testfornull интерпретатор Java переходит сюда.
```

Если оператор break прерывает цикл; то оператор continue завершает текущую итерацию цикла и начинает новую. Оператор continue, как с меткой, так и без нее, можно использовать только внутри цикла while, do или for. Оператор continue без метки заставляет ближайший цикл начать новую итерацию. Оператор continue с. меткой, то есть с именем окружающего цикла, заставляет этот цикл начать новую итерацию. Например:

```
for(lnt i=0; i<data.length; i--) {    // Цикл по массиву data.
 if(dafa[i] == -1)                          // Вели значение отсутствует,
  continue;                                // начать следующую итерацию.
 process(dafa[i]),                          // Обработать значение.
}
```

2.6 Классы и объекты

2.6.1 Класс

Рассмотрим синтаксис объявления класса. Класс объявляется с помощью ключевого слова `class`:

```
class Имя_класса {
    функции и переменные класса
}
```

Принято названия класса задавать начинающимся с большой буквы.Это не обязательное правило, а полезное для распознавания идентификаторов классов среди прочих идентификаторов (переменных, методов,пакетов).

Пример класса

```
/*
    Представляет точку в декартовом пространстве (x, y)
*/
public class Point {
 public double x, y;              // Координаты точки
 public Point (double x. double y) { // Конструктор, в котором
  this.x = x; this. y = y;        // инициализируются поля
 }
 public double distanceFromOrigin() {   //Метод, который оперирует
  return Math.sqrt(x*x + y*y);         // полями x и y
 }
}
```

Данное определение класса сохраняется в файле *Polnt.java* и компилируется в файл *Point.class.* После этого оно становится доступно для использования Java-программами и другими классами.

2.6.2 Объект

Объект класса объявляется следующим образом:

```
Имя_класса имя_объекта;
```

Для того чтобы создать экземпляр класса или ссылочного типа (выделить место в памяти) нужно использовать оператор `new`:

```
имя_объекта=new Имя_класса();
```

После этого фактически имя объекта ссылается на память выделенную для объекта.

Доступ к свойствам (методам и функциям) объекта класса теперь может осуществляться посредством оператора ‘.’:

```
Имя_класса.имя_объекта.имя_свойства
```

Пример объекта

```
// Создать объект Point с координатами (2, -3.5)
// и сохранить его в переменной p
Point p = new Point(2.0, -3.5);
p.y = p.x+p.x;
double d = p.distanceFromOrigin();
```

Доступ к библиотечным классам – пакетам осуществляется посредством этого же опеатора ‘.’:

```
имя_пакета.Имя_класса
```

Имя пакета `java.lang` может быть опущено и часто опускается. Пример этому – вызов метода `System.out.println()`.Класс `System` относится к пакету `java.lang`, `out` – объект,определенный в этом классе, класса `PrintStream`, `println()` является методом этого объекта.

Если объект используется в том же классе, в котором он объявлен, то при доступе к его свойствам не нужно указывать Имя_класса.

2.6.2 Введение управляющего класса.

Для тестирования созданого класса Point введем в программу управляющий класс

```
class Point
{
  int x, y;
  Point(int x, int y)
  {
    this.x = x;
    this. y = y;
```

```
  }
  double distance(int x, int y)
  {
    int dx = this.x - x;
    int dy = this.y - y;
    return Math.sqrt(dx*dx + dy*dy);
  }
  double distance(Point p)
  {
    return distance(p.x, p.y);
  }
}
class PointDist
{
  public static void main(String args[])
  {
    Point p1 = new Point(0, 0);
    Point p2 = new Point(30, 40);
    System.out.println("p1 = " + pi.x + ", " + p1.y);
    System.out.println("p2 = " + p2.x + ", " + p2.y);
    System.out.println("p1.distance(p2) = " + p1.distance(p2));
    System.out.println("p1.distance(60, 80) = " + p1.distance(60, 80));
  }
}
```

Как мы видим метод main перешел в управляющий класс PointDist и все объекты класс Point теперь создаются в нем.

3. Дополнительная литература

Java

1. **Мультимедийный Обучающий Курс TeachPro Java Для Начинающих**
2. Картузов А.В. **Программирование на языке JAVA**
3. Герберт Шилдт, Джеймс Холмс **Искусство программирования на JAVA**.
4. Патрик Нотон, Герберт Шилдт **Полный справочник по Java**.
5. Вязовик Н.А. **Программирование на JAVA. Курс лекций на intuit.ru**

4. Порядок выполнения работы

В соответствии с **вариантом** выполните следующее основное задание:

1. Создайте новый проект в среде Eclipse.
2. Создайте новый **класс 1**, который будет выполнять обработку массива.
3. Создайте новый **управляющий класс 2**, предусмотрев в нем точку входа (main).
4. Создать необходимые **приватные поля** в **классе 1** для работы с массивом. Сам массив инкапсулируется в класс 1.
5. Создать методы акцессоры для приватных полей **класса 1**. В том числе для всего массива и отдельных его элементов.
6. Сгенерировать необходимые конструкторы в **классе 1** на основе полей (Без параметров, с параметрами)
7. Создать метод **класса 1,** осуществляющий **обработку массива**, в соответствии с вариантом ***задания***.
8. Создать метод print в **классе 1,** для вывода массива.
9. Переопределить метод finalize(), который будет выводит сообщение о том,что массив и объект класса уничтожены.
10. Переопределить метод toString()
11. Создать объекта **класса 1** в **управляющем классе 2**.
12. Продемонстрировать всю функциональность **класса 1** в методе main.
13. Подготовьте отчет о выполнении лабораторной работы:

Для успешной сдачи лабораторной работы необходимо:

- представить преподавателю отлаженный код программы для указанного варианта задания;
- подготовить отчет по работе.

5. Порядок оформления отчета

Отчет о выполнении лабораторной работы должен содержать:

1) титульный лист;
2) задание;
3) текст программы;
4) результаты работы программы.

6. Варианты заданий

Вариант	Задание
1	Выполнить сортировку массива целых чисел по возрастанию.
2	Найдите наименьший элемент в двумерном массиве и номер строки и столбца в котором они расположены.

3	Найдите сумму всех нечетных чисел массива целых чисел.
4	Класс моделирует бросание двух костей, используя функцию random()(класс Math пакета java.lang) генерировать результат бросания 2 костей. Затем должна подсчитываться сумма двух значений (каждая кость может показать целое значение от 1 до 6) Используйте одномерный массив, чтобы показать сколько раз выпадет каждая сумма. Ваша программа должна бросать кость 300 раз.
5	Дан массив вещественных чисел (инициализировать с помощью функции Math.random()) вывести порядковый номер того из них, которое наиболее близко к введенной пользователем цифре(использовать функцию System.in.read()).
6	Элементы одномерного массива циклически сдвинуть на два элемента влево.
7	Элементы одномерного массива циклически сдвинуть на пять элементов вправо.
8	Выполнить сортировку массива символов по алфавиту.
9	Выполнить сортировку массива вещественных чисел по убыванию.
10	Нечетные строки матрицы A заменить на ‘x’
12	Найти максимальный элемент в массиве целых чисел. Программа должна используя функцию random() для заполнения массива.
13	Найти минимальный элемент в массиве вещественных чисел. Программа должна используя функцию random() для заполнения массива.
14	Подсчитать количество отрицательных элементов в матрице. Программа должна используя функцию random() для заполнения
15	Подсчитать количество нулевых элементов в матрице. Программа должна используя функцию random() для заполнения