**Лабораторная работа №5**

**Основы программирования на Java: примитивные и ссылочные типы данных, литералы**

**Цель работы:** Освоить основные способы создания Java-программ либо с помощью обычного редактора, либо с помощью среды разработки. Приобрести навыки работы с примитивными типами данных в Java.

**Теоретические сведения**



Переменные не что иное, как зарезервированные места памяти для хранения значений. Это означает, что при создании переменной Вы резервируете некоторое пространство в памяти.

Основываясь на типе данных, который присвоен переменной, операционная система выделяет память и решает, что может быть сохранено в зарезервированную памяти. Поэтому, назначая различные типы данных для переменных, в Java можно хранить целые числа, десятичные дроби или символов в этих переменных.

Существует два типа данных в Java:

* простые или примитивные типы данных;
* ссылочные типы данных (ссылка/объект).

**Примитивные типы данных**

Есть **восемь типов данных**, поддерживаемых Java. Основные типы данных предопределены языком и названы по [ключевому слову](http://proglang.su/java/syntax#klyuchevye-slova-v-java). Теперь давайте посмотрим в деталях эти восемь базовых типов данных существующих в языке программирования Java.

**Тип byte**

Тип данных byte является 8-разрядным знаковым целым числом.

* Минимальная величина равна -128 (-27).
* Максимальное значение равно 127 (включительно) (27-1).
* По умолчанию – 0.
* byte предназначен для экономии места в больших массивах, главным образом вместо целых чисел, поскольку byte в четыре раза меньше, чем int.

Пример:

byte a = 100;

byte b = -50;

**Тип short**

Тип данных short является 16-разрядным знаковым целым числом.

* Минимальное значение равно -32768 (-215).
* Максимальная величина равна 32 767 (включительно) (215-1).
* Тип short в Java может также использоваться для экономии памяти как byte. Размер short в 2 раза меньше, чем int.
* По умолчанию — 0.

Пример:

short s = 10000;

short r = -20000;

**Тип int**

В языке Java тип данных int является 32-разрядным знаковым целым числом.

* Минимальный размер int — 2 147 483 648 (-231).
* Максимальная величина равна 2,147,483,647 (включительно)(231-1).
* Тип int обычно используется для целых значений. Если нет озабоченности по поводу памяти.
* По умолчанию равно 0.

Пример:

int a = 100000;

int b =-200000;

**Тип long**

Тип данных long является 64-разрядным знаковым целым числом.

* Минимальное значение равно — 9,223,372,036,854,775,808 (-263).
* Максимальная величина — 9,223,372,036,854,775,807 (включительно). (263-1).
* В Java Применяется когда требуется более широкий диапазон, чем int.
* По умолчанию — 0L.

Пример:

long a = 100000L;

long b =-200000L;

**Тип float**

Тип данных float является c одинарной точностью 32-битный IEEE 754 с плавающей точкой.

* Тип float используется главным образом для сохранения памяти в больших массивах чисел с плавающей точкой.
* По умолчанию — 0.0f.
* Тип float никогда не должен применяется для точного значения, например, валюты.

Пример:

float f1 = 234.5f;

**Тип double**

Тип данных double является c двойной точностью 64-битный IEEE 754 с плавающей точкой.

* Обычно используется для десятичных значений.
* Тип double никогда не должен применяется для точного значения, например, валюты.
* По умолчанию — 0.0d.

Пример:

double d1 = 123.4;

**Тип boolean**

Тип данных boolean представляет собой один бит информации.

* Существует только два возможных значения: true и false.
* Предназначен для простых признаков, которые позволяют отслеживать условия true или false.
* По умолчанию — false.

Пример:

boolean one = true;

**Тип char**

Тип данных char является одним 16-разрядным символом Юникода.

* Минимальная величина равна «\u0000» (или 0).
* Максимальная величина равна «\uffff» (или 65535 включительно).
* В Java char нужен для хранения любого символа.

Пример:

char letterA ='A';

**Ссылочные типы данных**

Ссылочные переменные создаются с использованием определенных конструкторов классов. Они предназначены для доступа к объектам. Эти переменные объявляются с определенным типом, который не может быть изменен. Например, Employee, Puppy и т.д.

Объекты класса и различные виды переменных массива подпадают под **ссылочный тип данных**.

По умолчанию в Java значение любой переменной ссылки - null.

Ссылочная переменная может применяться для обозначения любого объекта, объявленного или любого совместимого типа.

Пример:

Animal animal = new Animal("giraffe");

**Литералы в Java**

**Литерал** — представление исходного кода как фиксированное значение. Оно представлено непосредственно в коде без каких-либо вычислений.

Литерал в Java можно назначить любой переменной из основного типа. Например:

byte a = 68;

char a = 'A';

Byte, int, long, и short может выражаться десятичной (основание 10), шестнадцатеричной (основание 16) или восьмеричной (основание 8) системой исчисления.

При использовании литералов в Java, префикс 0 применяться для указания восьмеричной системы, а префикс 0x указывает на шестнадцатеричную систему. Например:

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

Строковые литералы в языке Java задаются, как и в большинстве других языков, заключив последовательность символов между парой двойных кавычек. Примеры строковых литералов:

"Hello World"

"two\nlines"

"\"This is in quotes\""

Типы литералов String и char могут содержать любые символы Юникода. Например:

char a = '\u0001';

String a = "\u0001";

Язык Java поддерживает несколько специальных управляющих последовательностей для литералов String и char:

|  |  |
| --- | --- |
| **Обозначение** | **Представление** |
| \n | Новая строка (0x0a) |
| \r | Возврат каретки (0x0d) |
| \f | Прогон страницы (0x0c) |
| \b | Возврат на шаг (0x08) |
| \s | пробел (0x20) |
| \t | Табуляция |
| \" | Двойная кавычка |
| \' | Апостроф |
| \\ | Обратная косая черта |
| \ddd | Восьмеричный символ (ddd) |
| \uxxxx | Шестнадцатеричный символ UNICODE (xxxx) |

**Типы переменных, объявление и инициализация**

Переменная предоставляется нам именем хранения, чтобы нашей программой можно было манипулировать. Каждая переменная в Java имеет конкретный тип, который определяет размер и размещение её в памяти; диапазон значений, которые могут храниться в памяти; и набор операций, которые могут быть применены к переменной.

Необходимо объявить все переменные, прежде чем их использовать. Ниже показана основная форма объявления:

тип данных переменная [ = значение], [переменная [= значение], ...] ;

Чтобы объявить более чем одну переменную указанного типа, можно использовать список с запятыми в качестве разделителей.

Ниже приведены примеры объявления переменной и инициализации в Java:

int a, b, c; // Объявление трех целых a, b, и c.

int a = 10, b = 10; // Пример инициализации.

byte b = 22; // Инициализация переменной b типа byte.

double pi = 3.14159; // Объявление и присвоение величины пи.

char a = 'a'; // Переменной a типа char присваивается значение 'a'.

Существует **три типа переменных**:

* локальные переменные;
* переменные экземпляра;
* статические переменные или переменные класса.

**Локальные переменные в Java**

Локальные переменные объявляются в методах, конструкторах или блоках.

Локальные переменные создаются, когда метод, конструктор или блок запускается и уничтожаются после того, как завершиться метод, конструктор или блок.

Модификаторы доступа нельзя использовать для локальных переменных.

Они являются видимыми только в пределах объявленного метода, конструктора или блока.

Локальные переменные реализуются на уровне стека внутри.

**В Java не существует для локальных переменных значения по умолчанию**, так что они должны быть объявлены и начальное значение должны быть присвоено перед первым использованием.

**Пример**

«age» — локальная переменная, определяется внутри метода «pupAge()» и области её действия ограничивается только этим методом.

public class Test{

public void pupAge(){

int age = 0;

age = age + 7;

System.out.println("Возраст щенка: " + age);

}

public static void main(String args[]){

Test test = new Test();

test.pupAge();

}

}

Будет получен следующий результат:

Возраст щенка: 7

**Пример без инициализации**

Пример использования «age» без инициализации. Программа выдаст ошибку во время компиляции.

public class Test{

public void pupAge(){

int age;

age = age + 7;

System.out.println("Возраст щенка: " + age);

}

public static void main(String args[]){

Test test = new Test();

test.pupAge();

}

}

Это приведёт к следующему сообщению об ошибке во время компиляции:

Test.java:4:variable number might not have been initialized

age = age + 7;

^

1 error

**Переменные экземпляра**

Переменные экземпляра объявляются в классе, но за пределами метода, конструктора или какого-либо блока.

Когда для объекта в стеке выделяется пространство, создается слот для каждого значения переменной экземпляра.

В Java переменные экземпляра создаются тогда, когда объект создан с помощью [ключевого слова](http://proglang.su/java/syntax#klyuchevye-slova-v-java) «new» и разрушаются тогда, когда объект уничтожается.

Переменные содержат значения, которые должны ссылаться более чем на один метод, конструктор или блок, или на основные части состояния объекта, которые должны присутствовать на протяжении всего класса.

Переменные экземпляра могут быть объявлен на уровне класса, до или после использования.

Модификаторы доступа могут быть предоставлены для переменных экземпляра.

Переменные экземпляра в Java являются видимыми для всех методов, конструкторов и блоков в классе. Как правило рекомендуется сделать их private (уровень доступа). Однако можно сделать их видимыми для подклассов этих переменных с помощью модификаторов доступа.

Переменные экземпляра имеют значения по умолчанию. Для чисел по умолчанию равно 0, для логических — false, для ссылок на объект — null. Значения могут быть присвоены при объявлении или в конструкторе.

Переменные экземпляра в Java могут быть доступны непосредственно путем вызова имени переменной внутри класса. Однако в статических методах и различных класса (когда к переменным экземпляра дана доступность) должны быть вызваны используя полное имя — *ObjectReference.VariableName*.

**Пример**

import java.io.\*;

public class Employee{

// Переменная экземпляра открыта для любого дочернего класса.

public String name;

// Переменная salary видна только в Employee.

private double salary;

// Имя переменной присваивается в конструкторе.

public Employee (String empName){

name = empName;

}

// Переменной salary присваивается значение.

public void setSalary(double empSal){

salary = empSal;

}

// Этот метод выводит на экран данные сотрудников.

public void printEmp(){

System.out.println("имя: " + name );

System.out.println("зарплата:" + salary);

}

public static void main(String args[]){

Employee empOne = new Employee("Олег");

empOne.setSalary(1000);

empOne.printEmp();

}

}

Программа будет производить следующий результат:

имя: Олег

зарплата: 1000.0

**Переменные класса или статические переменные в Java**

**Переменные класса, также известные в Java как статические переменные**, которые объявляются со статическим ключевым слово в классе, но за пределами метода, конструктора или блока.

Там будет только одна копия каждой статической переменной в классе, независимо от того, сколько объектов создано из него.

Статические переменные или переменные класса в Java используются редко, кроме когда объявляются как константы. Константы - переменные, которые объявлены как public/private, final и static. Константы никогда не меняются от первоначального значения.

В Java статические переменные создаются при запуске программы и уничтожаются, когда выполнение программы остановится.

Видимость похожа на переменную экземпляра. Однако большинство статических переменных объявляются как public, поскольку они должны быть доступны для пользователей класса.

Значения по умолчанию такое же, как и у переменных экземпляра. Для чисел по умолчанию равно 0, для данных типа Boolean — false; и для ссылок на объект — null. Значения могут быть присвоены при объявлении или в конструкторе. Кроме того, они могут быть присвоены в специальных блоках статического инициализатора.

Статические переменные могут быть доступны посредством вызова с именем класса *ClassName.VariableName*.

При объявлении переменных класса как public, static, final, имена находятся в верхнем регистре. Если статические переменные такими не являются, синтаксис такой же, как у переменных экземпляра и локальных.

**Пример**

import java.io.\*;

public class Employee{

// salary(зарплата) переменная private static

private static double salary;

// DEPARTMENT(отдел) — это константа

public static final String DEPARTMENT = "Разработка ";

public static void main(String args[]){

salary = 1000;

System.out.println(DEPARTMENT+"средняя зарплата: "+salary);

}

}

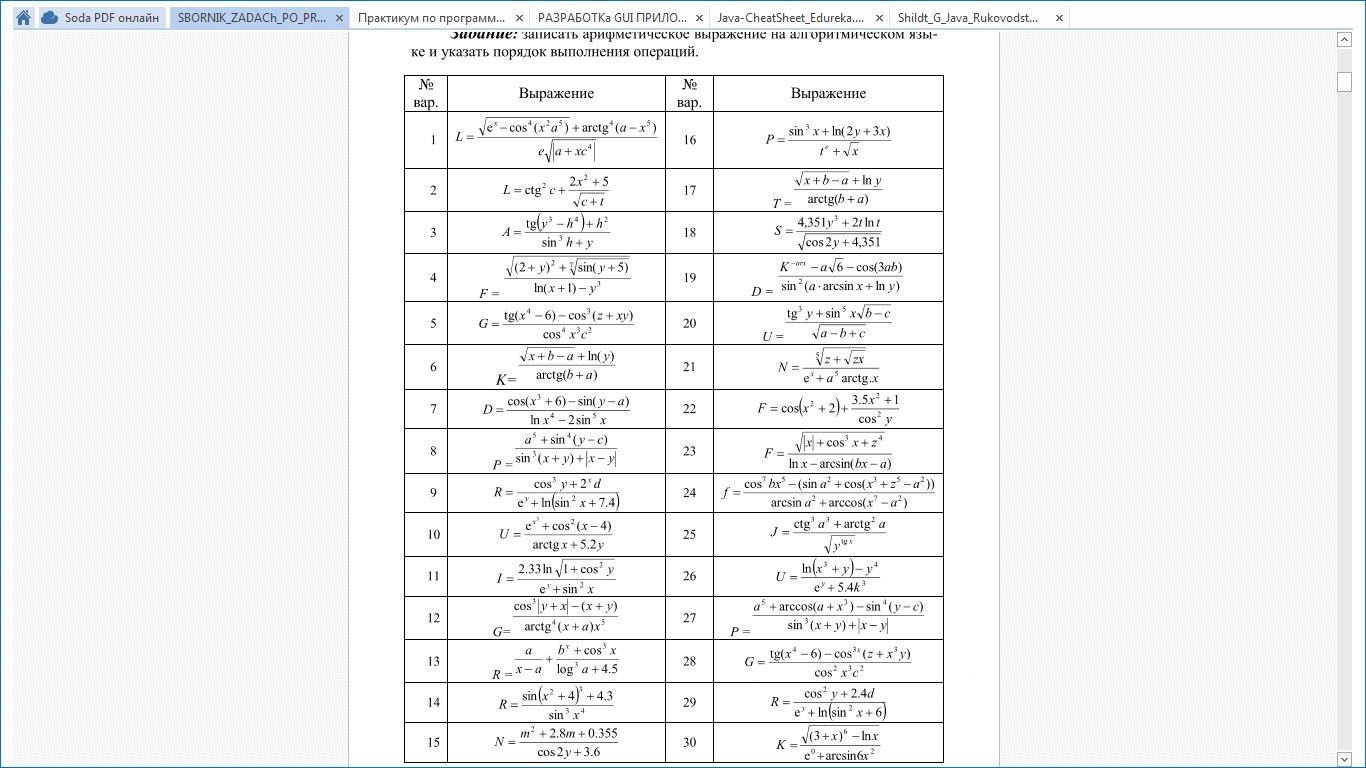
Это будет производить следующий результат:

Разработка средняя зарплата: 1000

**Примечание:** для получения доступа из внешнего класса, константы должны быть доступны как Employee.DEPARTMENT.

**Задание**

Изучить теоретический материал и выполнить задачи по вариантам.



**Форма отчета**

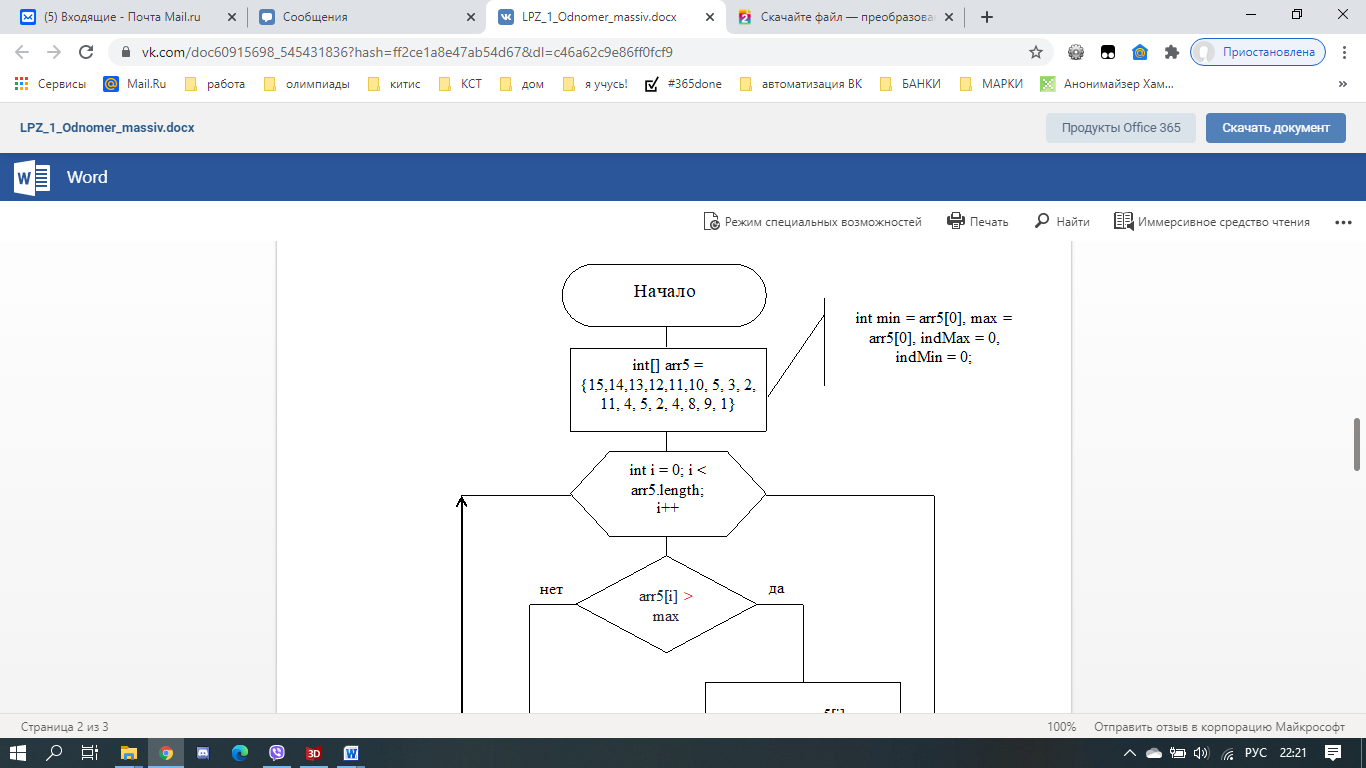
1. Тема, цель, индивидуальное задание

2. Таблица идентификаторов

|  |  |  |
| --- | --- | --- |
| Наименование переменной | Тип данных | Назначение |
|  |  |  |
|  |  |  |
|  |  |  |

3. Блок-схема алгоритма (см. Приложение)

Пример:



4. Код программы

5. Скриншот результата

**Контрольные вопросы для защиты**

1. Что такое переменная? Константа?
2. Правила именования.
3. Правила инициализации.
4. Примитивные типы данных в Java.

**Приложение**

**Условные графические обозначения в схемах алгоритмов и программ, отображающие основные операции процесса обработки данных и программирования по ГОСТ 2.708 - 81**

Размер, а следует выбирать из ряда 10, 15, 20 мм. Допускается увеличение размера а на число, кратное 5. Размер b принима­ют равным 1,5а.

|  |  |
| --- | --- |
|  | Процесс. Выполнение операций (группы операций), в результате которых изменяется значение (форма представления, расположение) данных  Решение. Выбор направления алгоритма (програм­мы) в зависимости от некоторых переменных условий |
|  | Решение. Выбор направления алгоритма (програм­мы) в зависимости от некоторых переменных условий |
|  | Модификация. Выполнение операций, меняющих команды (группы команд), изменяющих программу |
|  | Предопределенный процесс. Использование ранее созданных и описанных отдельно алгоритмов (программ) |
|  | Ввод-вывод. Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод) |
|  | Соединитель. Указание связи между прерванны­ми линиями потока (связывющие символы) |
|  | Пуск-останов. Начало, конец, прерывание про­цесса обработки данных или выполнения программы |
|  | Межстрочный соединитель |