**Лабораторная работа №3**

**Основы программирования на Java**

**Цель работы:** Освоить основные способы создания Java-программ либо с помощью обычного редактора, либо с помощью среды разработки. Используя примеры программ познакомиться с основными приемами в программировании на Java. Приобрести навыки в использовании системы помощи для поиска нужной информации по различным классам Java.

**Теоретические сведения**



В первое время при появлении Java двумя основными формами Java-программ являлись **приложение** и **апплет**. В данной работе рассматриваются различия между этими программами и их назначение на основе создания простейшего кода, как для приложения, так и для апплета.

Java-программы могут выполняться под управлением специального интерпретатора (java.exe), работающего в рамках отдельного процесса, либо под управлением браузера или навигатора Интернет, такого, как Microsoft Internet Explorer или Netscape Navigator. В последнем случае программа называется апплетом.

**Java-приложения** работают независимо от навигатора, главное их отличие от апплетов лежит в их назначении. Приложения похожи на программы, созданные, например, с использованием языка C/C++, хотя для своей работы они требуют присутствия среды Java – Java виртуальной машины (JVM). Но, в отличие от апплетов, их существование никак не связано с Internet и они не выполняются как содержимое страниц WWW. Это полноправные приложения, которые существуют и выполняются в локальных компьютерных системах пользователей.

**Java-апплеты** же разработаны для функционирования в сети и выполняются как часть станиц WWW, поэтому к ним относятся как к исполняемому содержимому. Хотя они и встраиваются в страницы WWW подобно стандартному содержимому, созданному с использованием HTML, на самом деле это программы, которые запускаются и выполняются, позволяют организовать интерактивное взаимодействие с пользователем.

Апплеты требуют наличия соответствующего Java-браузера, так как они должны загружаться по сети с сервера WWW в обеспечивающую их работоспособность среду исполнения Java на локальном компьютере.

***Простейшее приложение***

Перед созданием приложения познакомимся с последовательностью действий для создания приложений:

*Использование JDK (Java Developer’s Kit).*

1. Создание, ввод и сохранение обычного тестового файла, содержащего код программы, имеющего расширение .java (например, Hello.java). Использовать можно любой текстовый редактор, позволяющий работать с файлами, имеющими длинные имена, например Notepad.
2. Компиляция исходного кода Java в машинный байтовый код при помощи компилятора javac. В результате трансляции создаются файлы с расширением .class (Hello.class).
3. Исполнение приложения: передача файла байтового кода интерпретатору java для выполнения приложения.

***Замечание*.** Для выполнения компиляции и запуска приложения можно создать командный файл (с расширением .bat) следующего содержания:

*javac.exe Hello.java*

*java.exe Hello*

**Использование среды разработки**

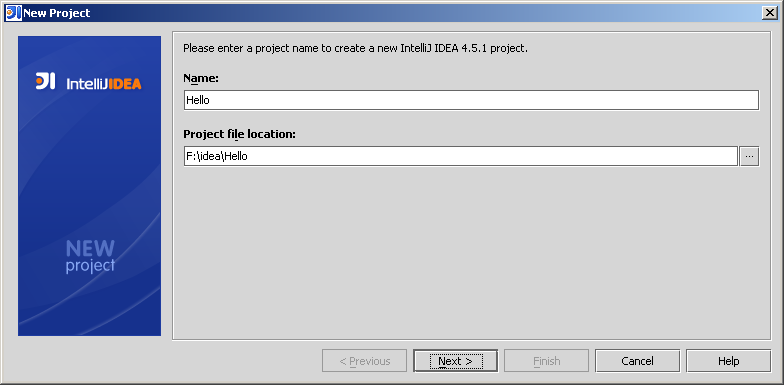
Для создания программ на Java возможно использование любой из сред разработки. Это может быть или ***Microsoft Visual J++, или JBuilder, или Idea, или Eclipse***. В последнее время наибольшую популярность получили две последние оболочки.

Порядок создания приложения или программы отличается в разных средах. Но в общем случае он подчиняется следующим правилам:

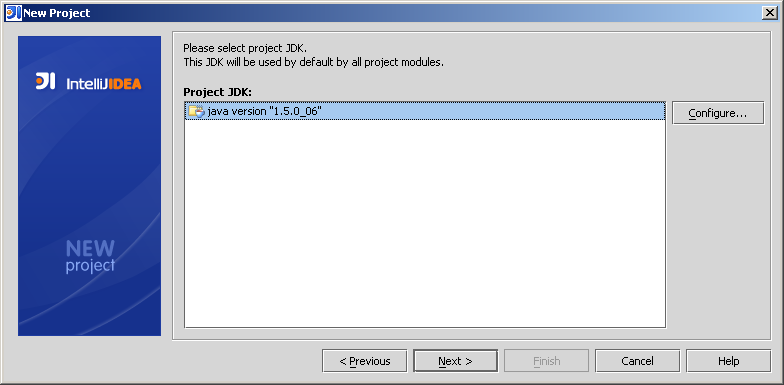
1. Создание нового Java-проекта.
2. Создание нового файла для класса или классов, ввод исходного текста при помощи встроенного в среду разработки текстового редактора.
3. Компиляция исходных текстов.
4. Запуск приложения. Обычно при создании приложения необходимо указать класс, который содержит главный метод main.

*Использование среды IntellijIdea*

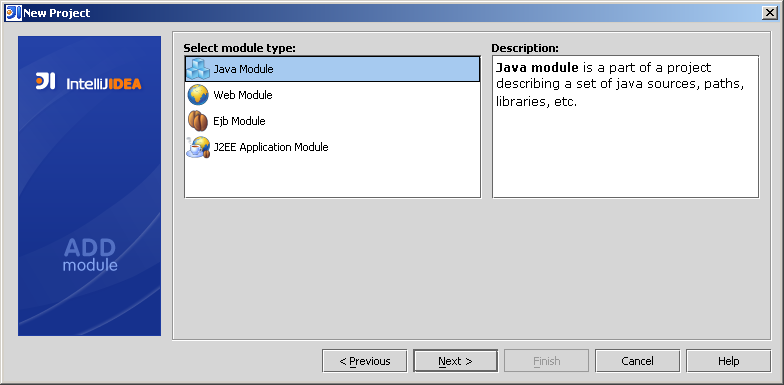
1. Создание нового Java-проекта. (File – New Project)



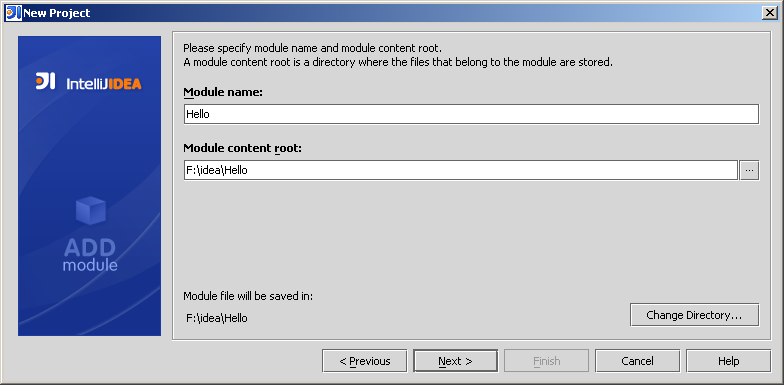
1. Указание JDK для построения приложения



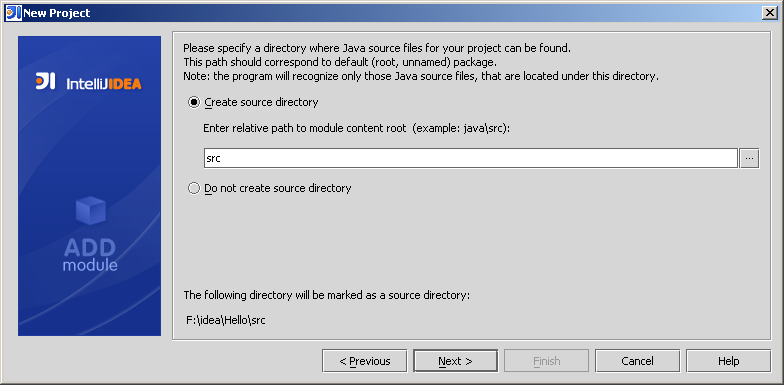
1. Выбор модуля для создания



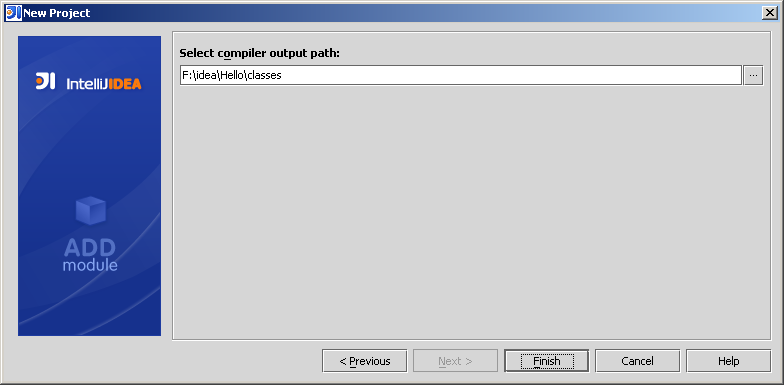
1. Указать имя модуля и папку для него



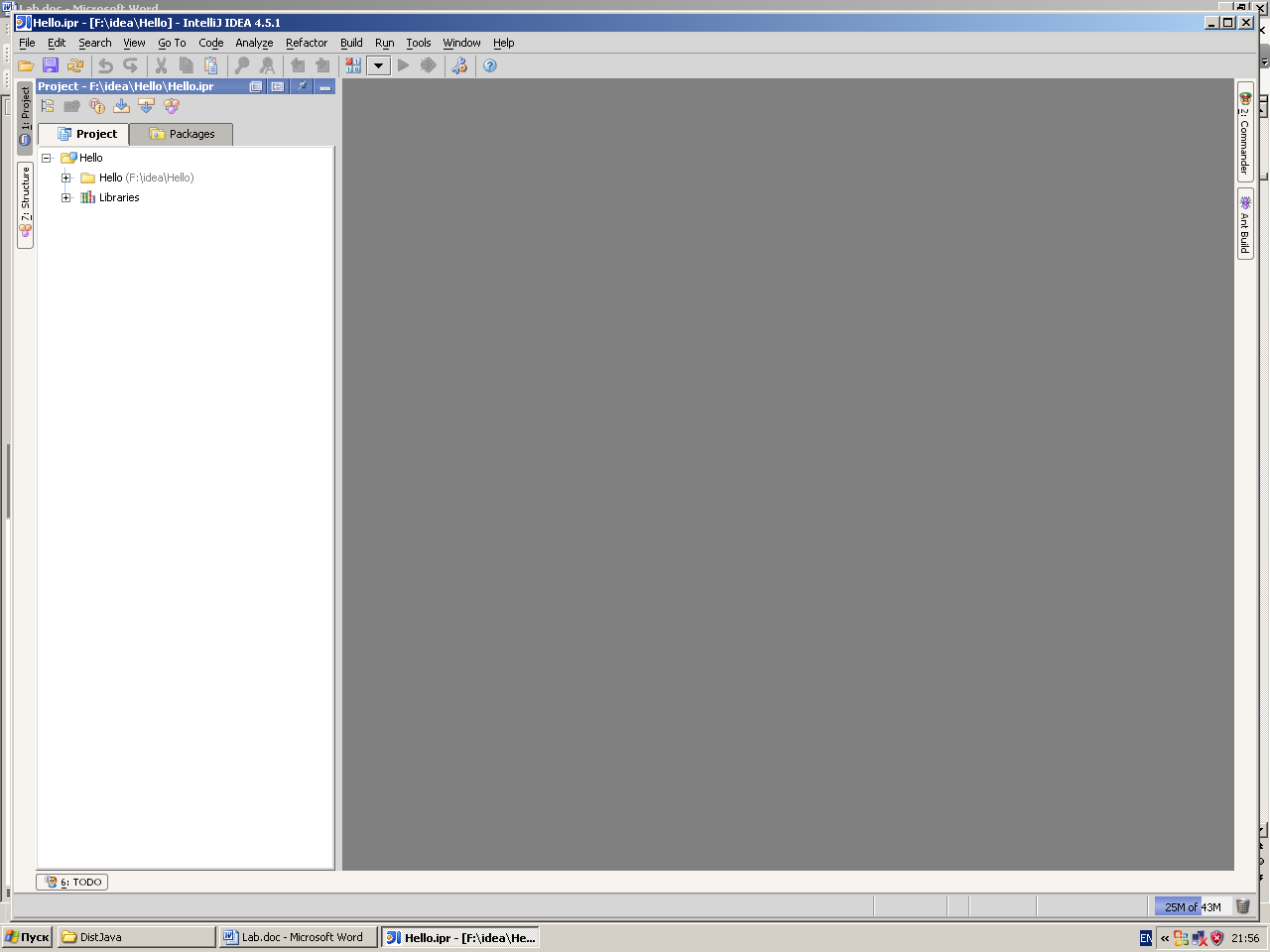
1. Указать папку для расположения исходных файлов



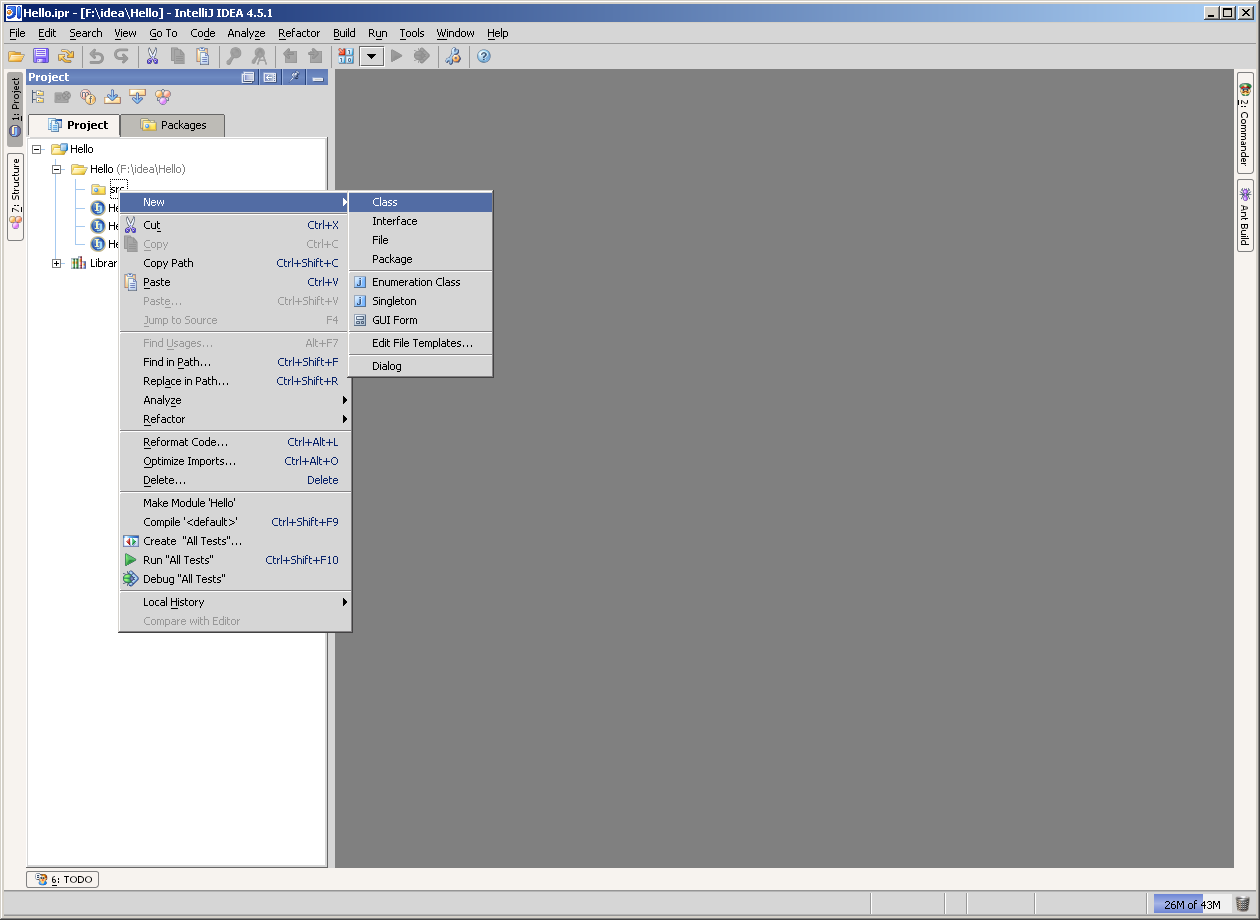
1. Указать папку для откомпилированных классов



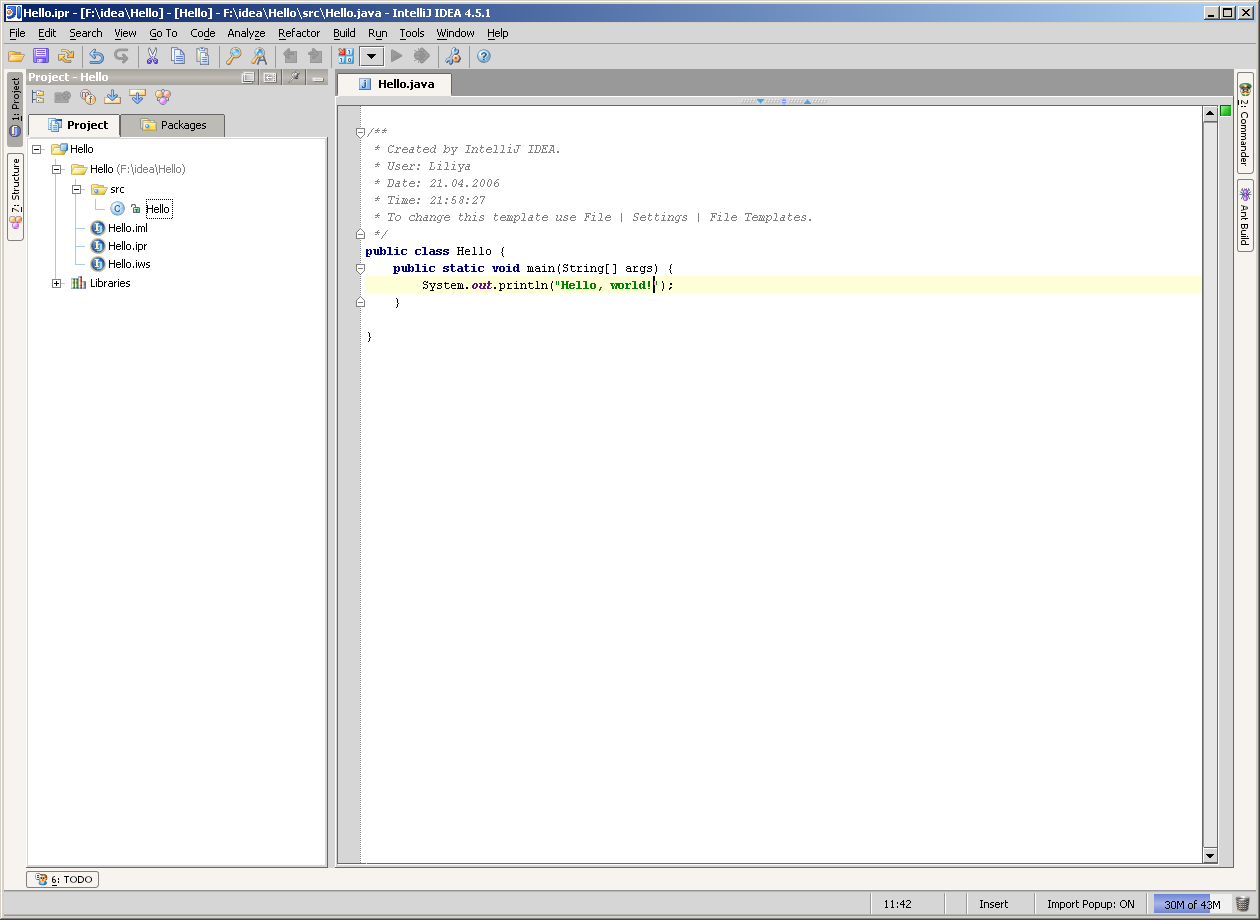
1. Просмотр содержимого проекта – закладка Project



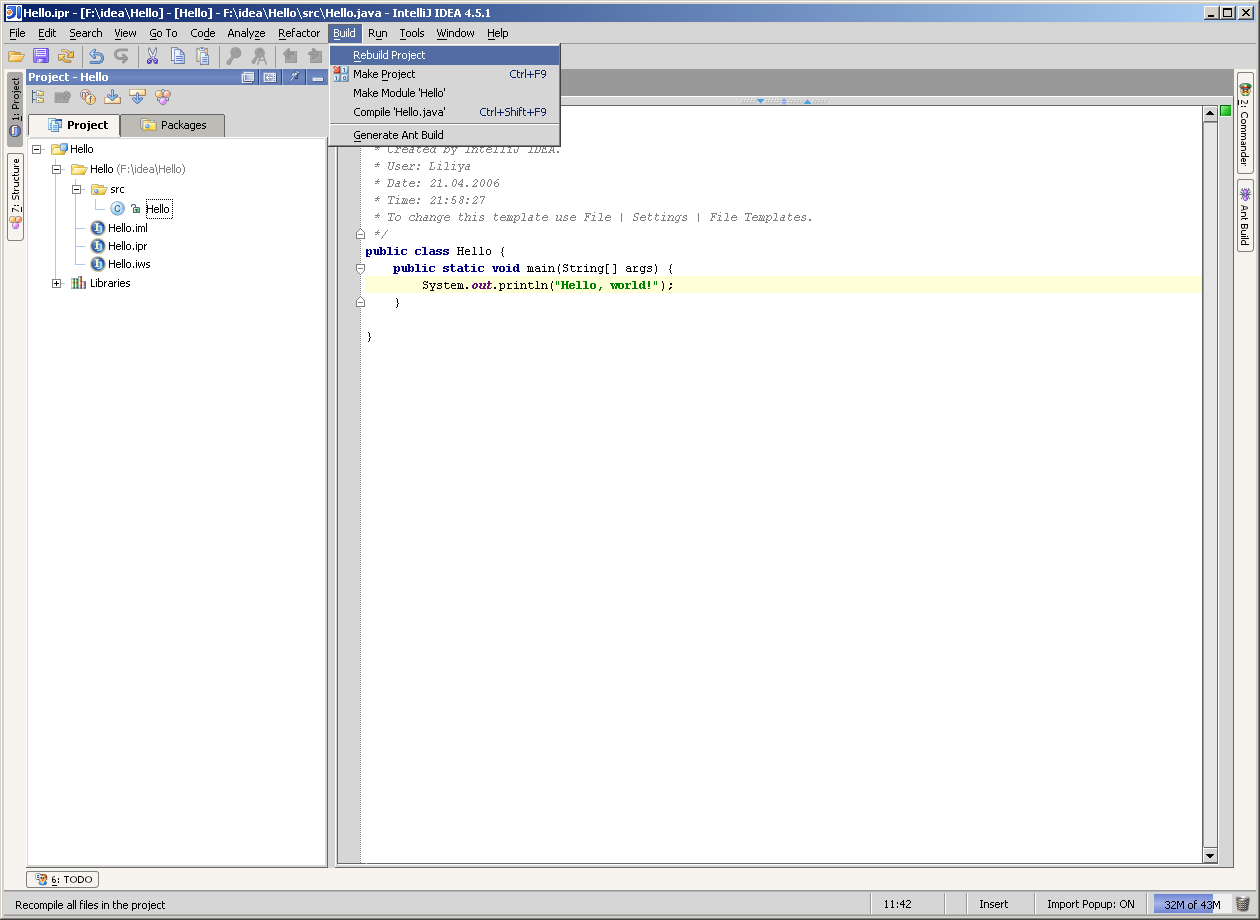
1. Создание нового класса (правая кнопка мыши на папке src)



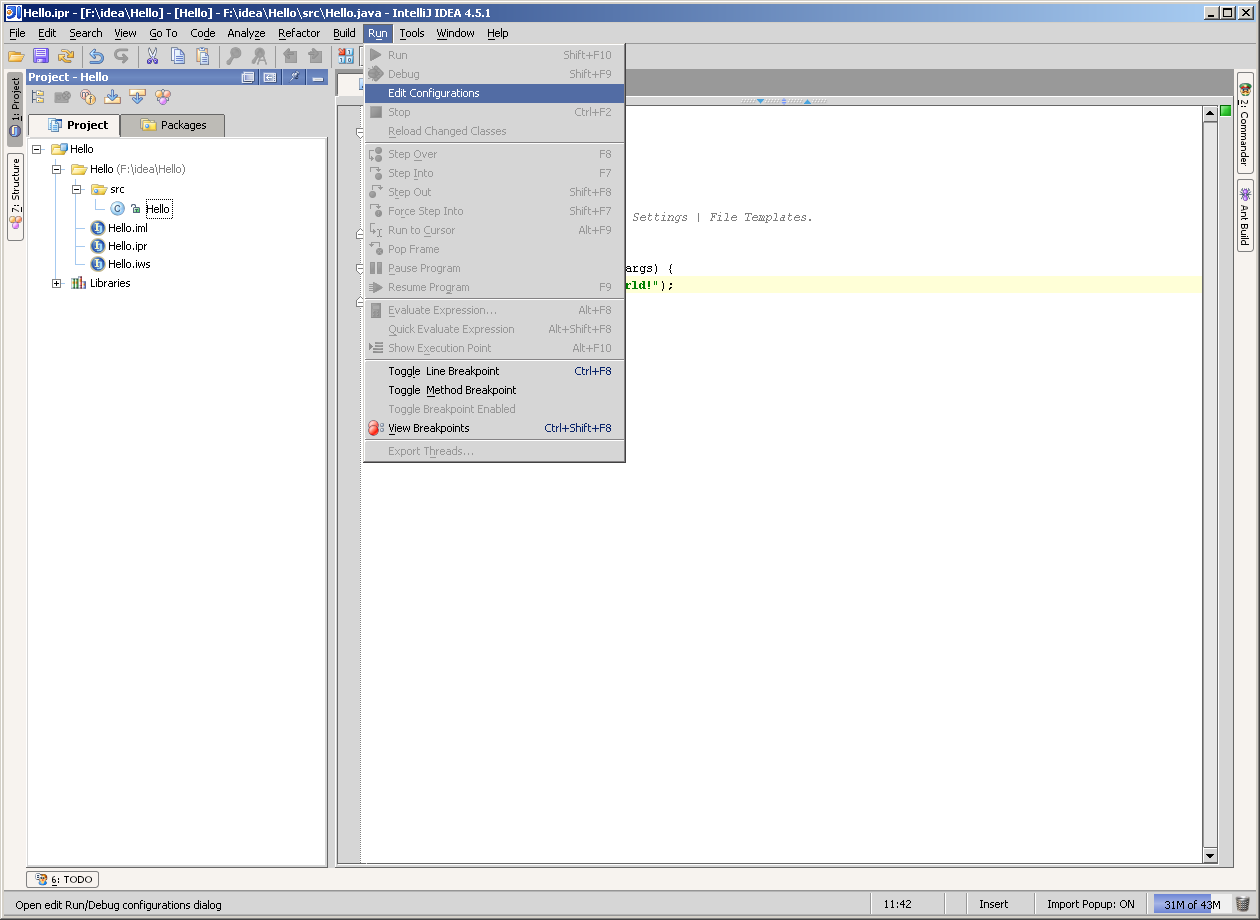
1. Составление текста класса



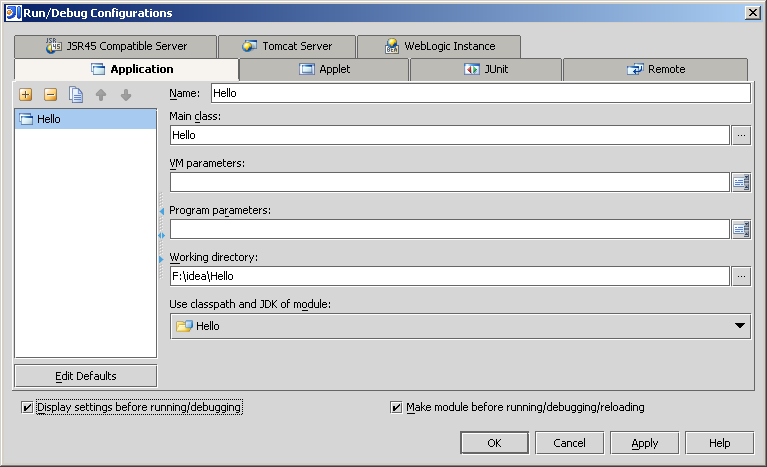
1. Построение проекта



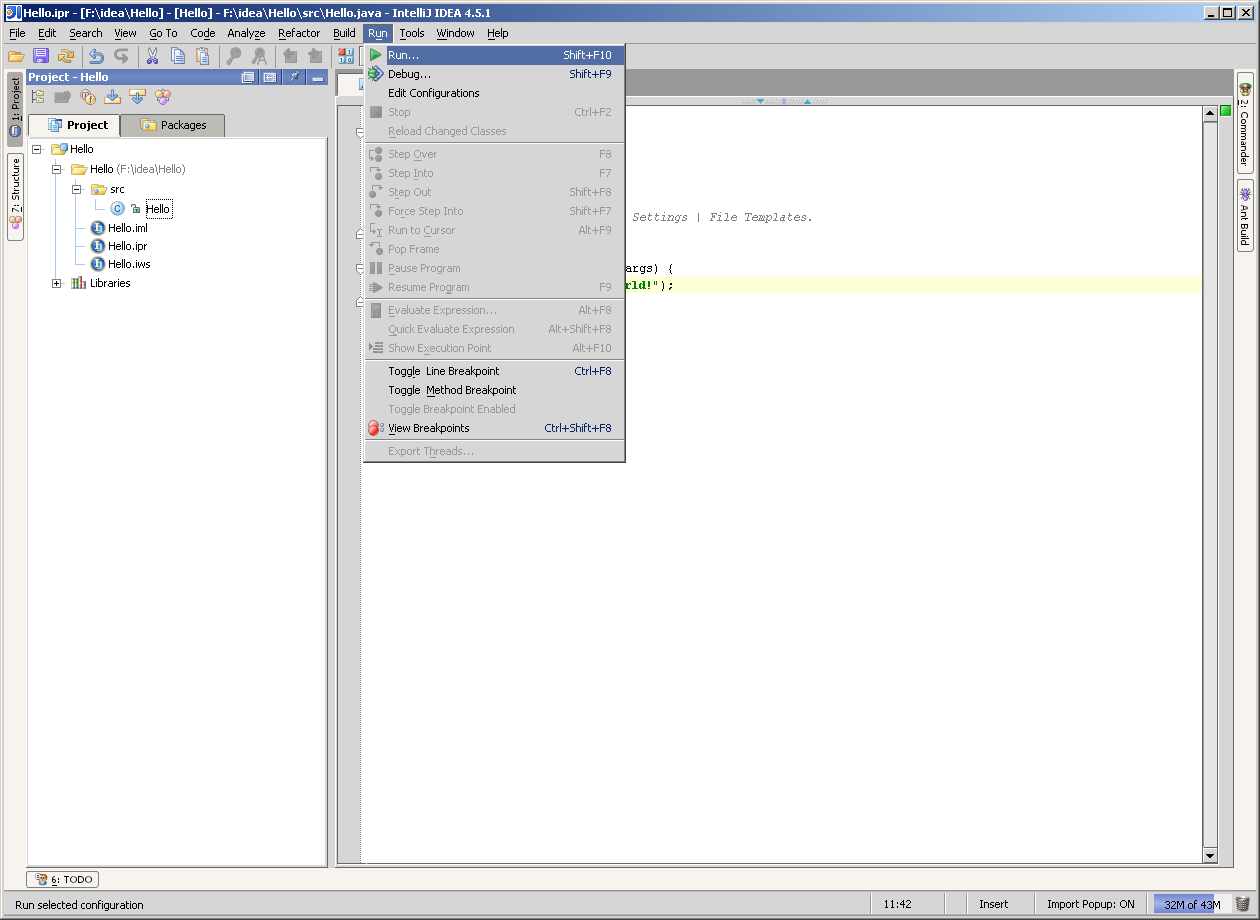
1. Создание конфигурации



12. Указание класса с методом main



13.Запуск приложения



**Приложение Hello**

Приступим к созданию простейшего приложения. Java-файл должен содержать следующий код:

/\*------------- Пример 1. Файл Hello.java -------------\*/

import java.util.\*;

**public class Hello {**

**public static void main(String args[]) {**

System.out.println("Hello, world");

Date d=new Date();

System.out.println("Date:"+d.toString());

}

} /\*---------------------------------------------------------------\*/

Так как класс Hello объявлен как public, то имя файла, в котором содержится его исходный код, должно совпадать с именем класса. Для классов, не объявленных как public, имена содержащих их исходные тексты файлов могут быть любыми (расширение обязательно .java).

Рассмотрим текст приложения примера 1. В строке

public **class** Hello {

объявляется новый **класс**, Hello, тело которого начинается с открывающей фигурной скобки и заканчивается закрывающей фигурной скобкой в последней строке исходного текста. На основе класса создаются объекты типа Hello.

Java - объектно-ориентированный язык, в котором основными строительными блоками являются объекты. В Java все построено на классах и объектах. Язык Java не поддерживает глобальные функции и глобальные переменные, а это означает, что все определяется из классов. Класс содержит весь код **состояния**(данные) и **поведения** (методы) объектов.

Все классы являются производными (или **подклассами**) от существующих классов. В случае класса Hello не было явно указано, от какого он класса он произошел. В таком случае - если не определен **суперкласс,** то по умолчанию предполагается, что таким суперклассом является **Object**. Для того, чтобы явно задать суперкласс, используется ключевое слово **extends**, например:

*public* ***class*** *Hello* ***extends*** *Object {*

В классе Hello объявляется метод **main()** со строковым параметром args, который будет содержать аргументы командной строки, передаваемые при запуске приложения:

*public static void* ***main****(String args[]) {*

Подобно языку C/C++, в приложение должна быть включена функция main(). Без нее интерпретатор не сумеет понять, откуда начинать выполнение приложение (метод main() является точкой входа приложения). И точно также, как в языках C/C++, Java-приложения могут запускаться **с аргументами командной строки**. Хотя необходимо обязательно включать параметр args в определение метода main(), но использовать аргументы командной строки необязательно. Ключевые слова **public** и **static**, называемые **модификаторами доступа,** рассматриваются ниже в разделе, посвященном классам.

Метод main() выводит две строки, одна из них “Hello, world”, вторая содержит текущую дату. Метод **System.out.println()** позволяет выводить информацию на экран. Этот вывод используется обычно в приложениях для текстового вывода, в апплетах же используется графический вывод.

Если методу System.out.println() передать строку символов, заключенную в пару двойных кавычек, этот метод выведет данную строку на экран, завершив ее переводом строки. Кроме того, этот метод можно использовать для печати значений переменных - как по отдельности, так и со строками символов, например:

*System.out.println("Symbol array");*

*int i=7; System.out.println(i);*

*int j=10; System.out.println("j="+i);*

**Структура Java-программы**

Познакомимся с основными блоками Java-программы. Все рассматриваемые в данном разделе приложения не относятся к апплетам, но все сказанное ниже в равной степени применимо и к апплетам.

Все Java-программы содержат в себе четыре разновидности блоков: **классы** (classes), **методы** (methods), **переменные** (variables) и **пакеты** (package).

*Методы* есть не что иное, как функции или подпрограммы. В *переменных* же хранятся данные. Данные понятия присутствуют так или иначе во всех языках программирования. С другой стороны, *классы* представляют собой фундамент объектно-ориентированных свойств языка. Для простоты на данном этапе изучения языка Jаva можно сказать, что класс - это некое целое, содержащее переменные и методы.

Наконец, *пакеты* содержат в себе классы и помогают компилятору найти те классы, которые нужны ему для компиляции пользовательской программы. Классы, входящие в один пакет, особым образом зависят друг от друга, пока же, опять-таки для простоты, пакеты можно рассматривать просто как наборы классов. Например, приложение Hello импортирует пакет java.util, в котором содержится класс Date.

Вышеперечисленные блоки присутствуют в любой Java-программе. Но программы могут включать в себя составные части и других видов (**интерфейсы, исключения, потоки**), которые не требуются обязательно для каждой программы, однако во многих программам без них не обойтись.

Java-программа может содержать в себе любое количество классов, но один из них всегда имеет особый статус и непосредственно взаимодействует с оболочкой времени выполнения (**первичный класс**). В таком классе обязательно должны быть определены один (для приложений) или несколько (для апплетов) специальных методов. Для приложений первичный класс должен обязательно содержать метод main().

Рассмотрим подробнее каждый из основных блоков Java-программы: переменные, методы, классы и пакеты.

***Переменные***

*Переменную* можно представить как хранилище для единицы данных, имеющее собственное имя. Любая переменная принадлежит определенному *типу*. Тип переменной определяет, какую информацию в ней можно хранить. Переменные должны быть объявлены с использованием следующего синтаксиса (модификаторы рассматриваются ниже в разделе, посвященном классам):

***<Модификаторы> ТипПеременной ИмяПеременной ;***

В Java существует два вида переменных. Первый - **примитивные типы**(primitive types). К ним относятся стандартные, **встроенные в язык** типы для представления численных значений, одиночных символов и булевских (двоичных, логических) значений. Все примитивные типы имеют **предопределенный размер** занимаемой ими памяти. Ко второму виду переменных - **ссылочные типы** (reference type) - относятся типы, определенные пользователем (классы, интерфейсы) и типы массивов. Все ссылочные типы являются **динамическими** типами, для них выделяется память во время работы программы.

Примитивные и ссылочные типы также различаются по тому, как переменные этих типов передаются в качестве параметров методам (то есть функциям). Переменные примитивных типов передаются **по значению**, тогда как ссылочные переменные всегда передаются **по ссылке**.

Практически самым важным различием между двумя типами переменных является то, что память для ссылочных переменных выделяется динамически, во время выполнения программы. Использование переменных ссылочных типов требует явного запрашивания требуемого количества памяти для каждой переменной прежде, чем можно будет сохранить в этой переменной какое-либо значение. Причина проста - оболочка времени выполнения сама по себе не знает, какое количество памяти требуется для того или иного ссылочного типа. Рассмотрим в качестве примера использования переменных разных типов следующую программу:

/\*------------- Пример 2. Файл VarTypes.java -------------\*/

class VarTypes

{ public static void main(String args[])

{

// переменная примитивного типа

int varPrimitive;

// сразу после объявления можно в нее записывать данные

varPrimitive=1;

// переменная ссылочного типа

int varReference[]; // или: int[] varReference;

// выделение памяти для переменной этого типа

varReference=new int[3];

// теперь можно сохранять данные в переменной этого типа

varReference[0]=2;

varReference[1]=3;

varReference[2]=4;

System.out.println("varPrimitive="+varPrimitive);

System.out.println("varReference[0]="+varReference[0]);

System.out.println("varReference[1]="+varReference[1]);

System.out.println("varReference[2]="+varReference[2]);

}

}

/\*--------------------------------------------------------/

Так как тип int относится к примитивным типам, оболочка выполнения знает, сколько места необходимо выделить такой переменной (четыре байта). Однако при объявлении *массива* переменных типа int, оболочка выполнения не может знать, сколько места потребуется для хранения этого массива. Поэтому прежде, чем разместить что-либо в переменной ссылочного типа, необходимо запросить у системы определенное количество памяти под эту переменную. Этот запрос осуществляется с помощью оператора **new**.

Переменные-массивы и другие ссылочные переменные лишь **указывают** на то место в памяти, где содержатся собственно данные, тогда как переменные примитивных типов ни на что не указывают, а просто **содержат** в себе соответствующие данные, имеющие определенный фиксированный размер.

Ссылочные переменные, хоть и очень похожи на указатели C/C++, имеют сильное отличие от них. Используя ссылочные типы, нельзя получить доступ к фактическим адресам данным в памяти.

Примитивные типы

Всего в Java определено восемь примитивных типов: **int** (4b), **short** (2b), **byte** (1b), **long** (8b), **float** (4b), **double** (8b), **boolean** (true, false), **char** (2b).

Первые шесть типов предназначены для хранения численных значений, с ними можно производить арифметические операции. Тип char предназначен для хранения символов в стандарте Unicode.

Булевские (логические, двоичные) переменные могут иметь **одно из двух допустимых значений**: true или false. Двоичные константы являются именно константами, а не строками. Их нельзя преобразовать в строковый тип. Они также не являются целыми значениями нуль или единица, как двоичные значения в языках C/C++. Рассмотрим примеры использования булевских переменных:

*boolean b1=true, b2, b3;*

*System.out.println(b1); // печать: true*

*b2=(25==25); // b2 равно true*

*b2=(25==10); // b2 равно false*

*b3=(b1&b2); // b3 равно false*

*System.out.println("b1&b2="+b3); // печать: b1&b2=false*

*b3=(b1!=b2); // b3 равно true*

*System.out.println("b1 не равно b2 - "+b3);*

*// печать: b1 не равно b2 - true*

Булевским переменным можно присваивать не только булевские константы, но и результаты сравнения переменных различных типов. Операции !, !=, == работают с булевскими значениями так же, как одноименные операторы работают с целочисленными значениями в языке C/C++.

***Ссылочные типы***

Ссылочные типы отличаются от примитивных тем, что они не определены в самом языке Java, и поэтому количество памяти, требуемое переменных этих типов, заранее знать невозможно. Пример одного из ссылочных типов - это тип массива. Массивы языка Java могут состоять из переменных любых типов, включая типы, определенные пользователем (классы).

Познакомимся с некоторыми терминами, относящимся к работе с переменными ссылочных типов. Когда для переменной ссылочного типа выделяется память при помощи оператора new, то тем самым этот ссылочный тип **реализуется.** Таким образом, каждая переменная ссылочного типа является **реализацией, объектом** или **экземпляром** соответствующего типа.

Язык Java не позволяет просто объявить переменную ссылочного типа и сразу же начать записывать в нее значение. Необходимо сначала запросить у оболочки времени выполнения некоторый объем памяти, а оболочка, в свою очередь, должна сделать запись в своих внутренних таблицах, что активизирована переменная данного ссылочного типа. Весь этот процесс в целом и называется *реализацией* переменной. После реализации, когда уже имеется экземпляр переменной данного типа, можно использовать этот экземпляр для хранения данных. Важно понимать, что экземпляр переменной и сам ссылочный тип, к которому эта переменная относится, являются качественно различными понятиями - для хранения переменной можно использовать только реализованный экземпляр переменной ссылочного типа.

***Типы, определенные пользователем***

Язык Java позволяет определять **новые типы помощью новых классов**, а также с помощью интерфейсов (речь о них пойдет позже). Для простоты можно сказать, что классы похожи на структуры (или записи) языка С - они тоже позволяют хранить наборы переменных разных типов. Но в отличие от структур, классы помимо переменных могут хранить и методы.

Рассмотрим пример определения и использования нового класса (нового типа) MyType:

/\*------------- Пример 3. Файл NewClass.java -------------\*/

/\*------------- 3.1. Объявление нового типа -------------\*/

class MyType // объявляется класс

{ public int myData=5; // переменная-элемент класса

public void myMethod() // метод класса

{ // печать в методе

System.out.print("myMethod!");

System.out.println(" myData="+myData);

}

MyType() // конструктор без параметров

{ // печать в конструкторе

System.out.println("Constructor without parameters");

}

MyType(int v) // конструктор с одним параметром

{ // печать в конструкторе

System.out.print("Constructor with one parameter");

System.out.println(" Setting myData="+v);

myData=v;

}

}

/\*-- 3.2. Реализация объектов и действия с ними -------------\*/

class NewClass // первичный класс

{ public static void main(String args[])

{ // объект obj1 - реализация класса MyType

MyType obj1=new MyType();

obj1.myMethod(); // использование метода

// доступ к открытой переменной

System.out.println("---------obj1.myData="+obj1.myData);

// объект obj2 - реализация класса MyType

MyType obj2=new MyType(100);

// доступ к открытой переменной

System.out.println("---------obj2.myData="+obj2.myData);

}

}

/\*--------------------------------------------------------\*/

В части 3.1 определяется новый тип данных, а часть 3.2 иллюстрирует три основных вида действий, которые можно производить с объектом: *создание* объекта (реализация класса), доступ к *переменным-элементам* и доступ к *методам* этого объекта (через оператор “точка” (.)).

Поскольку тип myType является ссылочным типом, то для реализации класса необходимо использовать оператор new. Этот оператор запрашивает память у системы для хранения объекта. Кроме того, можно определить, какие еще действия должны выполняться в момент реализации класса, определив так называемый **конструктор** (constructor) - метод, имеющий такое же имя как и класс**.** В классе может быть не один конструктор, они должны отличаться списком аргументов.

Конструкторы можно использовать для инициализирующих действий, например для присвоения начальных значений. Можно использовать все определенные в классе конструкторы (с параметрами и без них), вызывая их при создании объектов класса. Если в классе отсутствует определение конструктора без параметров, то при использовании следующего оператора

*MyType obj1=new MyType();*

вызывается конструктор без параметров по умолчанию.

Нужно отметить, что название “типы, определенные пользователем” не подразумевает, что каждый пользователь *сам* должен определять для себя все типы. В состав интерфейса прикладного программирования (Application Programming Interface - API) входят десятки стандартных классов, которые можно использовать в своих программах.

***Класс String (тип строковых переменных)***

Данный класс представляет собой гибрид примитивных и ссылочных типов. В основе своей тип String является типом, определенным пользователем, так как он определяется как одноименный класс String, содержащий в себе методы и переменные. Но в то же время этот тип проявляет некоторые свойства примитивного типа, что выражается, в частности, в том, как осуществляется присвоение значение этим переменным при помощи знака операции = (но можно для инициализации создаваемых объектов пользоваться и явным вызовом конструктора), например:

// 1-ый способ инициализации строковой переменной

String S1="Hello";

// 2-ый способ инициализации строковой переменной

String S2=new String("Hello");

Кроме того, строковый тип проявляет свойства примитивных типов в том, что для конкатенации (сложения) двух строк можно использовать знак операции +, например:

String S0="Variable ";

int myInt=3;

// 1-ый способ инициализации строковой переменной

String S1=S0+"myInt"+myInt;

// 2-ый способ инициализации строковой переменной

String S2=new String(S0+"myInt"+myInt);

Несмотря на поддержку таких операций с примитивными типами как = и +, строковые переменные типа String являются в то же время и объектами, так что для них можно вызывать методы класса String, например, узнать длину строки:

int len=S1.length();

Итак, реализация переменных типа String не обязательно требует применения оператора new. Однако при программировании необходимо всегда помнить о том, что тип String является особым - это **единственный класс, переменные которого могут объявляться и использоваться без применения оператора new**.

***Типы массива***

Типы массива используются для определения массивов - упорядоченного набора однотипных переменных. Можно определить массив над любым существующим типом, включая типы, определенные пользователем. Кроме того, можно пользоваться и массивами массивов или многомерными массивами.

Иначе говоря, если можно создать переменную некоторого типа, значит можно и создать массив переменных этого типа. **Создание массивов требует использования оператора new**, например:

// объявление ссылочной переменной типа массива

int intArray[];

// реализация или создание массива переменных целого типа

intArray=new int[3];

// объявление ссылочной переменной типа массива

MyType objArray[];

// реализация или создание массива ссылочных переменных типа MyType

objArray=new MyType[3];

Оператор new запрашивает для массива необходимую память. При создании ссылочной переменной типа массив не нужно объявлять размер массива, это делается при реализации типа массива при помощи оператора new. Доступ же к элементам массива ничем не отличается от доступа к элементам массива в C/C++:

// доступ к элементам

intArray[0]=1; intArray[1]=2; intArray[2]=3;

// доступ к переменным и реализация объектов типа MyType

objArray[0]=new MyType();

objArray[1]=new MyType();

objArray[2]=new MyType();

// доступ к элементам объектов, объекты - элементы массива

objArray[0].myData=0; objArray[0].myMethod();

objArray[1].myData=0; objArray[1].myMethod();

objArray[2].myData=0; objArray[2].myMethod();

Массивы Java имеют три важных преимущества перед массивами других языков. **Во-первых,** программисту не обязательно указывать размер массива при его объявлении. **Во-вторых,** любой массив в языке Java является переменной - а это значит, что его можно передать как параметр методу и использовать в качестве значения, возвращаемого методом. И **в-третьих**, не составляет никакого труда узнать, каков размер данного массива в любой момент времени через специальную переменную **length**, имеющуюся в каждом массиве, например:

int len=intArray.length;

***Методы***

Метод представляет собой подпрограмму, аналогичную функциям языка C и Pascal, или методу класса в С++. Каждый метод имеет тип возвращаемого значения и может вызываться с передачей некоторых параметров.

Объявление метода имеет следующий синтаксис (модификаторы рассматриваются ниже в разделе, посвященном классам):

***<Модификаторы> ТипВозврЗначения ИмяМетода(<Параметры>)***

***{***

***ТелоМетодаСодержащееОбъявлениеПеременных&Операторы***

***}***

С каждым методом должен быть соотнесен **тип возвращаемого им значения**. Например, тип **void** является специальным способом указать системе, что данный метод **не возвращает** никакого значения. Методы, у которых возвращаемое значение принадлежит к любому другому типу, кроме void, должны содержать в своем теле оператор **return**. Возвращаемое значение может принадлежать к любому из типов, включая примитивные типы и ссылочные типы.

В качестве **параметров** в языке Java можно передавать переменные любого типа, включая типы, определенные через классы, и массивы переменных любого типа и размера. Однако в качестве параметров переменные примитивных типов ведут себя иначе, чем переменные ссылочных типов.

Все переменные примитивных типов передаются методам **по значению** (by value). Это означает, что в момент вызова метода делается копия переменной, передаваемой методу. Если метод в своем теле будет изменять значение переданной ему в качестве параметра переменной, то содержимое исходной переменной изменяться не будет, так как все действия будут производится с ее **копией**.

Напротив, значения переменных ссылочного типа, переданных в качестве параметров, **можно изменить** в теле метода. Когда методу в качестве параметра передается переменная ссылочного типа, то при изменении ее значения явным образом меняется содержимое того, на что указывает эта переменная.

*Замечание.* Несмотря на то, что тип строковых переменных (тип String) является определенным пользователем типом, он не ведет себя как ссылочный тип при передаче параметров. **Переменные типа String в качестве параметров всегда передаются по значению**, - то есть метод, получив строковую переменную в качестве параметра, в своем теле будет фактически работать с копией этой строковой переменной. Иначе говоря, изменение строковой переменной в теле метода не влияет на значение этой же переменной снаружи метода.

Иногда возникает необходимость создавать две или несколько функций, выполняющих по сути, одни и те же действия, но имеющие различные списки параметров. В языке Java можно присвоит **одно и то же имя** нескольким методам, которые различаются списками своих параметров. Этот процесс называется **совмещением или перегрузкой методов**.

Выгоды перегрузки методов особенно очевидны в том, что вместо запоминания нескольких имен разных методов можно ограничиться запоминанием одного только имени, общего для всех методов с разными параметрами. В обязанности компилятора входит выяснение того, какой именно метод требуется вызвать в каждом случае.

Приведем пример, в котором показаны: методы, которые возвращают значение, и не возвращают никакого значения; методы, работающие с переменными различных видов в качестве параметром; перегруженные методы. Все методы и переменная класса объявлены как static для того, чтобы можно было ими пользоваться без создания объекта класса.

/\*------------- Пример 4. Файл TestMethods.java -------------\*/

class TestMethods

{

static int v=0;

// функция, не возвращающая значение

static void setV(int i)

{ v=i;

System.out.println("Void method!");

}

// фунция с возвращаемым значением

static int getV()

{ System.out.println("Returning method!");

return v;

}

// функция для проверки работы с параметрами

static int func(int a, int b[])

{ a=a+1;

b[0]=b[0]+1;

System.out.println("a="+a+" b[0]"+b[0]); // a=2 b=2

return a; // return 2

}

// сравнение двух целых

static String compare(int i,int j)

{ if(i==j) return ""+i+" and "+j+" are equal";

else

if(i>j) return ""+i+" greater than "+j;

else return ""+j+" greater than "+i;

}

// совмещение метода с другим числом параметров

static String compare(int i,int j,int k)

{ String S="";

S=S+compare(i,j)+'\n';

S=S+compare(i,k)+'\n';

S=S+compare(j,k);

return S;

}

// совмещение метода с параметрами другого типа

static String compare(double i,double j)

{ if(i==j) return ""+i+" and "+j+" are equal";

else

if(i>j) return ""+i+" greater than "+j;

else return ""+j+" greater than "+i;

}

// главный метод - точка входа

public static void main(String args[])

{ // вызов метода, не возвращающего значения

setV(5);

// вызов метода, возвращающего значение

int vv=getV();

// передача параметров: примитивные и ссылочные типы

int A; int B[]=new int[1];

A=1; B[0]=1;

System.out.println("A="+A+" B[0]="+B[0]); // A=1 B[0]=1

int aa=func(A,B);

System.out.println("aa="+aa);

System.out.println("A="+A+" B[0]="+B[0]); // A=1 B[0]=2

// вызов совмещенных методов

String S;

S=compare(2,5); System.out.println(S);

S=compare(3,1,6); System.out.println(S);

S=compare(1.5,2.1); System.out.println(S);

}

}

/\*------------------------------------------------------\*/

**Задание**

Проверить и объяснить работу всех приложений, рассматриваемых в данной работе. Должны быть созданы следующие приложения **Hello** (пример 1), **VarTypes** (пример 2), **NewClass** (пример 3), **TestMethods** (пример 4)

**Форма отчета**

1. Тема, цель

2. Скриншоты выполненных программ

3. Ответы на контрольные вопросы

**Контрольные вопросы**

1. Чем отличаются Java-приложения и Java-апплеты?
2. Какие основные составные части должны присутствовать в каждой Java-программе, их функции (назначение)?
3. Что такое первичный класс приложения? Какой обязательный метод он должен содержать?
4. Какие существуют виды переменных Java, чем они отличаются друг от друга?
5. Какие примитивные типы определены в Java, особенности булевского типа?
6. Что называется процессом реализации ссылочного типа?
7. Что делает конструктор класса? Должен ли он обязательно явно присутствовать в объявлении класса?
8. Какие существуют виды ссылочных типов?
9. Что такое типы, определенные пользователем?
10. Что такое стандартные типы, определенные пользователем?
11. В чем особенности строковых переменных?
12. Чем массивы Java отличаются от массивов других языков, их преимущества?
13. Как переменные различных видов передаются в качестве параметров методам?
14. Как ведут себя строковые переменные при передачи их в качестве параметров?