

## ЛАБОРАТОРНАЯ РАБОТА №1. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ECLIPSE. РАЗРАБОТКА ПРОГРАММ НА JAVA

**Цель работы:** получить практические навыки работы с интегрированной средой разработки Eclipse; научиться писать простейшую программу на языке Java; иметь представление о методе `main()`.

### 1.1 Методические рекомендации

#### 1.1.1 Общие сведения о языке Java

Java – полностью объектно-ориентированный язык программирования. В Java отсутствует понятие процедур. С помощью Java решаются различные задачи и тот же самый круг проблем, что и на других языках программирования. Java может использоваться для создания двух типов программ: приложений и апплетов. *Приложение* – программа, которая выполняется на компьютере, под его операционной системой. Приложения Java могут быть непосредственно выполнены с использованием интерпретатора Java. *Апплет* – небольшая программа, работающая с окнами, которые внедрены в страницу HTML. Чтобы выполнить Java-апплеты, нужна поддержка Java web-браузером, то есть Internet Explorer, Firefox, Chrome и т. д., или средство просмотра апплета.

Java – это интерпретируемый и компилированный язык программирования. Исходный текст (файлы с расширением `a.java`) компилируется компилятором Java (`javac`), который преобразовывает исходный текст в байт-код (файлы с расширением `a.class`). Цель проектировщиков Java состояла в том, чтобы разработать язык, посредством которого программист может писать код, который будет выполняться всегда, в любое время.

#### Простота

Проектировщики Java пытались разработать язык, который могли бы быстро изучить программисты. Также они хотели, чтобы язык был знаком большинству программистов для простоты перехода. С этой целью в Java проектировщиками было удалено множество сложных особенностей, которые существовали в C и C++. Особенности, такие как манипуляции указателя, перегрузка оператора и т. д., в Java не существуют.

Java не использует *goto*-инструкцию, а также не использует файлы заголовка. Конструкции подобно *struct* и *union* были удалены из Java.

#### Объектная ориентированность

В Java всё может быть объектом. Так, основное внимание здесь уделяется свойствам и методам, которые оперируют данными в нашем приложении, и нет концентрации только на процедурах. Свойства и методы вместе описывают состояние и поведение объекта. В Java мы будем наталкиваться на термин «метод» очень часто, с ним мы будем должны познакомиться. Термин «метод» используется для функций.

#### Распределенность

Java может использоваться для разработки приложений, которые работают на различных платформах, операционных системах и графических интерфейсах пользователя. Java предназначен также для поддержки сетевых приложений. Таким образом,

Java широко используется как инструмент разработки в среде, подобной Internet, где существуют различные платформы.

### **Устойчивость**

Java – язык со строгим контролем типов, так что требуется явное объявление метода. Java проверяет код во время трансляции и во время интерпретации. Таким образом, устраняются некоторые типы ошибок при программировании. Java не имеет указателей и соответственно арифметических операций над ними. Все данные массивов и строк проверяются во время выполнения, что исключает возможность выхода за границы дозволенного. Преобразование объектов с одного типа на другой также проверяется во время выполнения.

Автоматическая обработка исключений: в традиционных средах программирования программист должен был вручную распределять память, и в конце программы имел явное количество свободной памяти. Возникали проблемы, когда программист забывал освобождать память. В Java программист может не беспокоиться о проблеме, связанной с освобождением памяти. Это делается автоматически, поскольку Java обеспечивает обработку исключений для объектов, которые не используются.

Обработка исключений упрощает задачу обработки ошибок и восстановления.

### **Безопасность**

Вирусы – большая причина беспокойства в мире компьютеров. До появления Java программисты должны были сначала просмотреть файл перед загрузкой и выполнением. Но даже после этого они не были уверены в надежности файла. Также существует много специальных программ, о которых мы должны знать. Эти программы могут находить уязвимые данные нашей системы.

Java обеспечивает управляемую среду, в которой выполнена программа. Java никогда не предполагает, что код может быть безопасно выполнен. И так как Java – больше чем язык программирования, он обеспечивает несколько уровней контроля защиты. Со справкой этих уровней он гарантирует безопасную среду выполнения.

Первый уровень – это безопасность, обеспеченная языком Java. Свойства и методы описываются в классе, и к ним можно обратиться только через интерфейс, обеспеченный классом. Java не позволяет никаких операций с указателями и таким образом запрещает прямой доступ к памяти. Избегается переполнение массивов. Проблемы, связанные с безопасностью и мобильностью, скрыты.

На следующем уровне компилятор, прежде чем приступить к компиляции кода, проверяет безопасность кода и затем следует в соответствии с протоколами, установленными Java.

Третий уровень – это безопасность, обеспеченная интерпретатором. Прежде чем байт-код будет фактически выполнен, он является полностью укрытым верификатором.

Четвертый уровень заботится о загрузке классов. Загрузчик класса гарантирует, что класс не нарушает ограничения доступа, прежде чем он загружен в систему.

### **Независимость от структуры системы**

Код Java может быть выполнен на разных платформах. Это достигается при смешении трансляции и интерпретации.

1 Программы Java оттранслированы в байт-код компилятором (*байт-код* – это универсальный машинный код).

2 Байт-код выполняется интерпретатором (Виртуальная Машина Java). Интерпретатор должен выполнять байт-код для каждой аппаратной платформы. Байт-код выполняется на любой версии Виртуальной Машины Java.

### **Мобильность**

Независимость от платформы означает легкость переноса программы с одного компьютера на другой без каких-либо трудностей. Также Java – платформа независимая на обоих уровнях, то есть на первичном (исходном) и на вторичном уровне.

Java – это язык со строгим контролем типов, что означает, что необходимо объявлять тип для каждой переменной. И эти типы данных в Java одинаковы для всех платформ. Java имеет свои собственные библиотеки фундаментальных классов, которые облегчают запись кода для программиста, который может быть перемещен с одной машины на другую, без потребности перезаписи кода. Независимость от платформы на исходном уровне означает, что можно переместить исходный текст из одной системы в другую, компилируя код и работая в системе.

*Платформа независимая на вторичном уровне* означает, что откомпилированный двоичный файл может быть выполнен на различных платформах, не перетранслируя код, если они имеют Виртуальную Машину Java, которая функционирует как интерпретатор.

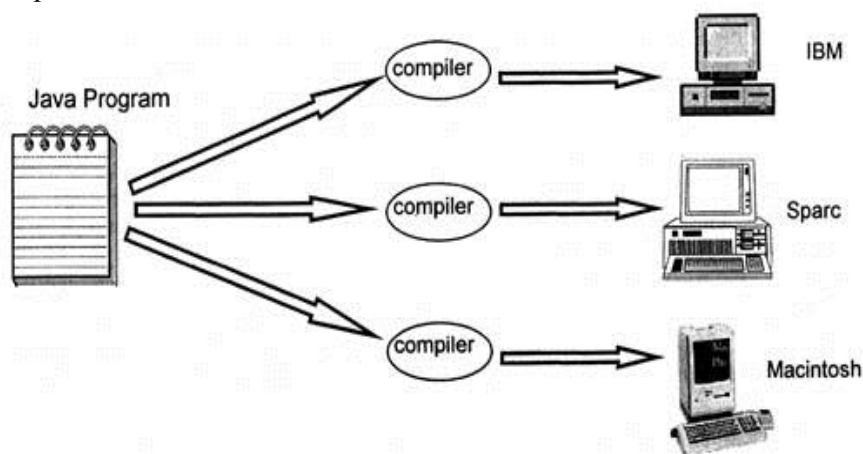


Рис. 1. Компилирование программ традиционным способом

Для программ, которые написаны на С и С++ или на любом другом языке, компилятор преобразует набор команды в машинный код или команды процессора. Эти команды являются специальными для процессора. В результате, если необходимо использовать этот код в некоторой другой системе, нужно найти компилятор для этой системы и откомпилировать код еще раз, так чтобы получился машинный код, определенный для этой машины. Рисунок 2 дает представление того, как программы Java могут быть выполнены на различных машинах.

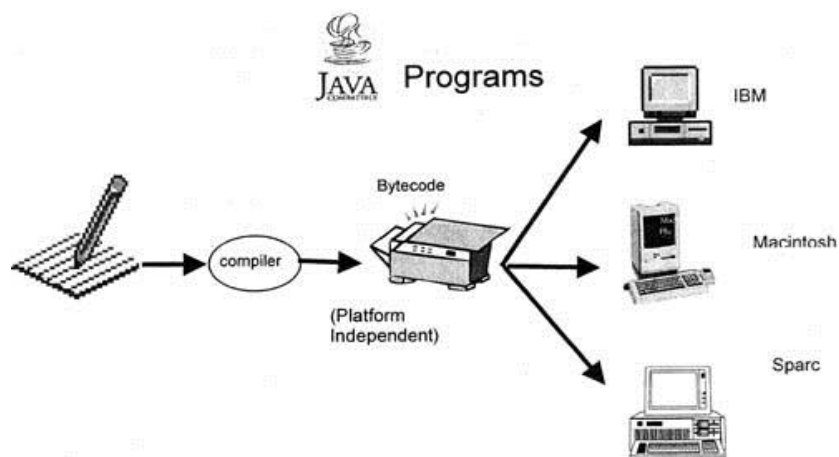


Рис. 2. Компилирование кода в Java

В отличие от C и C++, компилятор Java преобразует исходный текст в байт-коды, которые являются машинно-независимыми. Байт-коды – это только части команд Java, разрезанные на байты, которые могут быть декодированы любым процессором.

Интерпретатор Java, также именуемый как JVM (Виртуальная Машина Java), или Java Runtime Interpreter, выполняет байт-коды Java. Интерпретатор Java является частью среды разработки.

### 1.1.2 Средства разработки Java

**JDK** – средство разработки Java, используемое при разработке программ. Оно состоит из:

- классов;
- компилятора;
- отладчика;
- JRE (среды выполнения Java).

В JDK входят следующие *инструментальные средства*:

- `javac` – это команда, которая используется для компиляции исходного текста. Она конвертирует исходный файл (файл с расширением Java) в файл класса (файл с расширением `.class`);
- `java` – эта команда используется для выполнения файла класса, который исполняет классы в Виртуальной Машине Java;
- команда `appletviewer` позволяет выполнять апплеты без использования web-браузера.

### 1.2 Контрольный пример выполнения лабораторной работы

Для создания первой программы на языке Java запустите IDE Eclipse. В строке меню выберите *File*, пункт *New* и подпункт *Java project* (рис. 3).

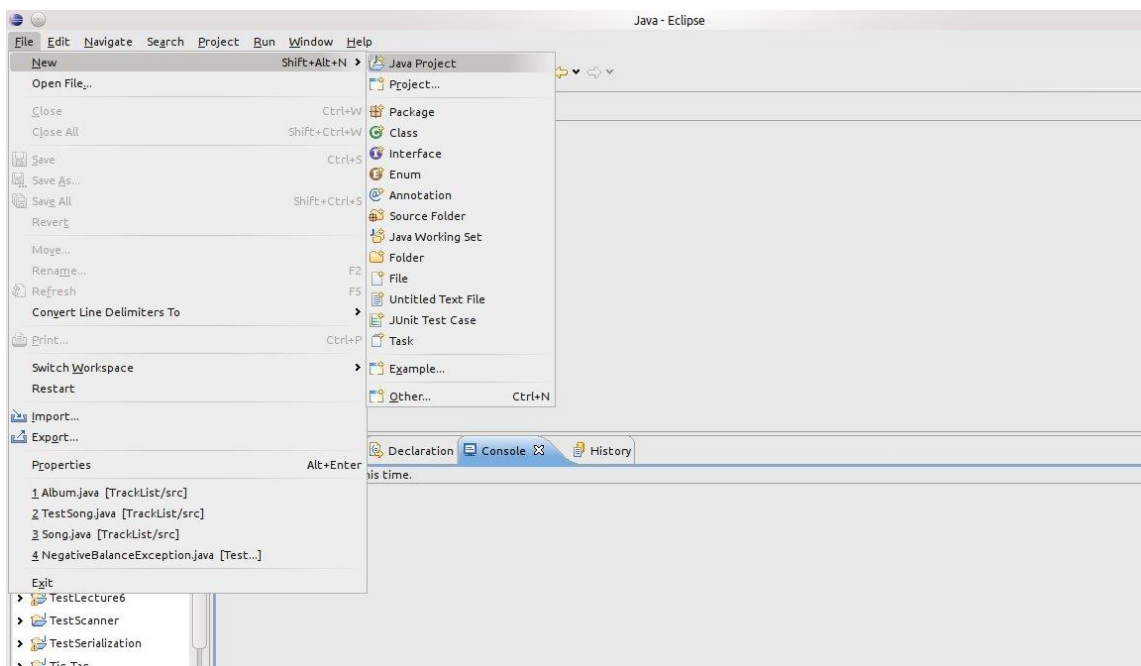


Рис. 3. Создание проекта в IDE Eclipse

В появившемся окне напишите имя создаваемого проекта, например *HelloWorld* (рис. 4)

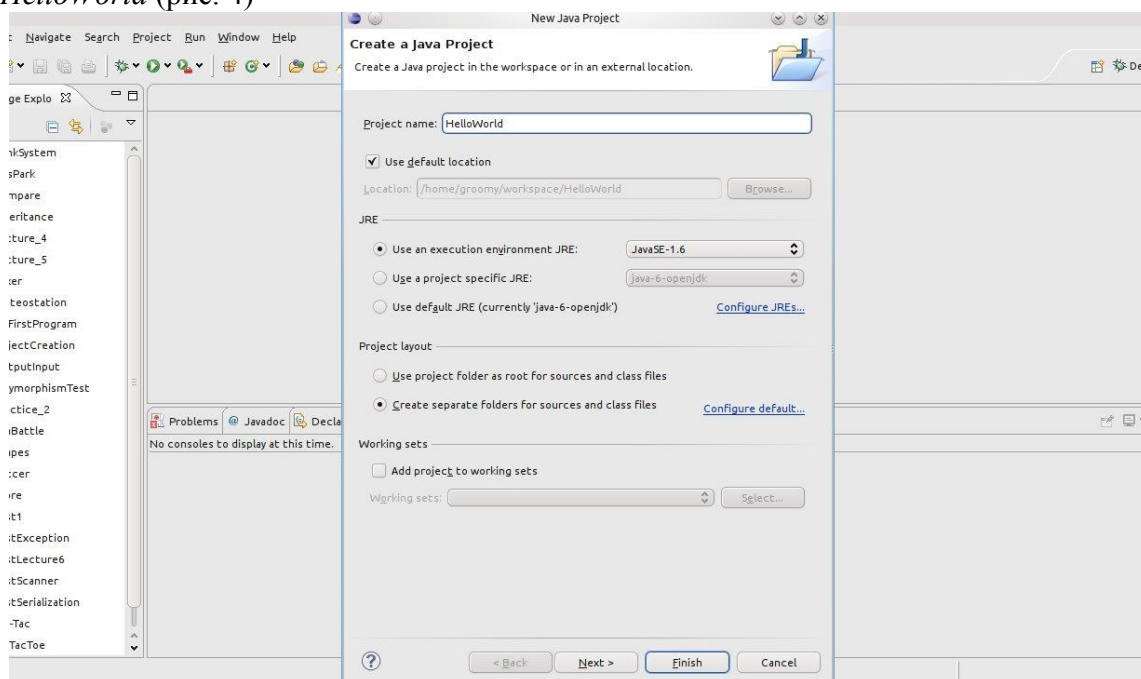


Рис. 4. Определение имени проекта в IDE Eclipse

В окне *Package Explorer* (находится слева) найдите созданный проект и нажатием правой кнопки мыши вызовите контекстное меню. В меню выберите пункты *New* → *Class* (рис. 5).

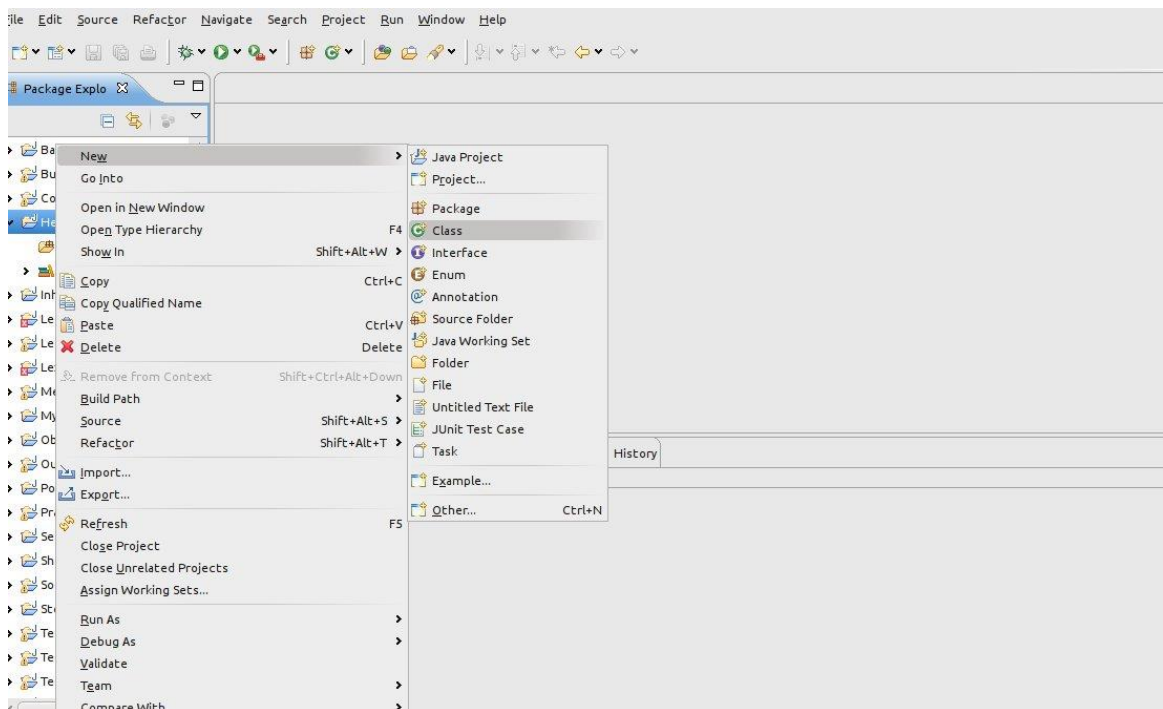


Рис. 5. Выбор меню создания нового класса в IDE Eclipse

Задайте имя класса *HelloWorld* в появившемся окне, уберите все галочки, если они выставлены, нажмите кнопку *Finish* для окончания создания класса (рис. 6).

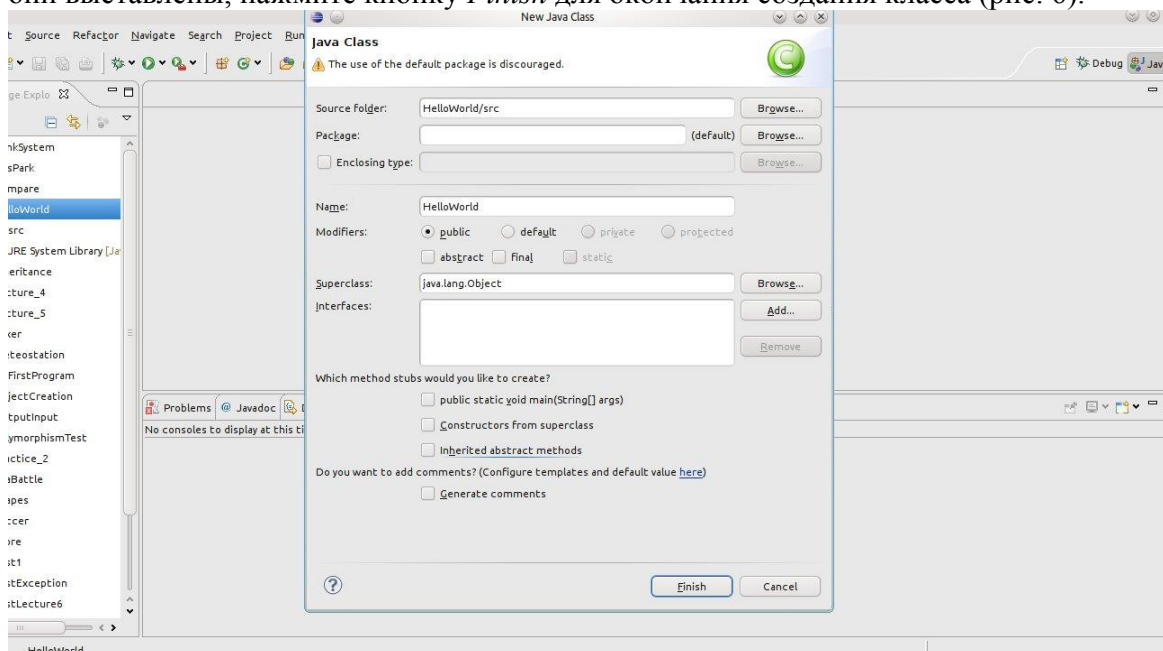


Рис. 6. Создание класса в IDE Eclipse

В появившемся окне редактирования наберите программу:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Для выполнения программы найдите файл в Package Explorer и, вызвав контекстное меню, выберите пункты Run as → *Java Application* (рис. 7).

Результат выполнения программы вы увидите в консоли (нижняя часть IDE Eclipse).

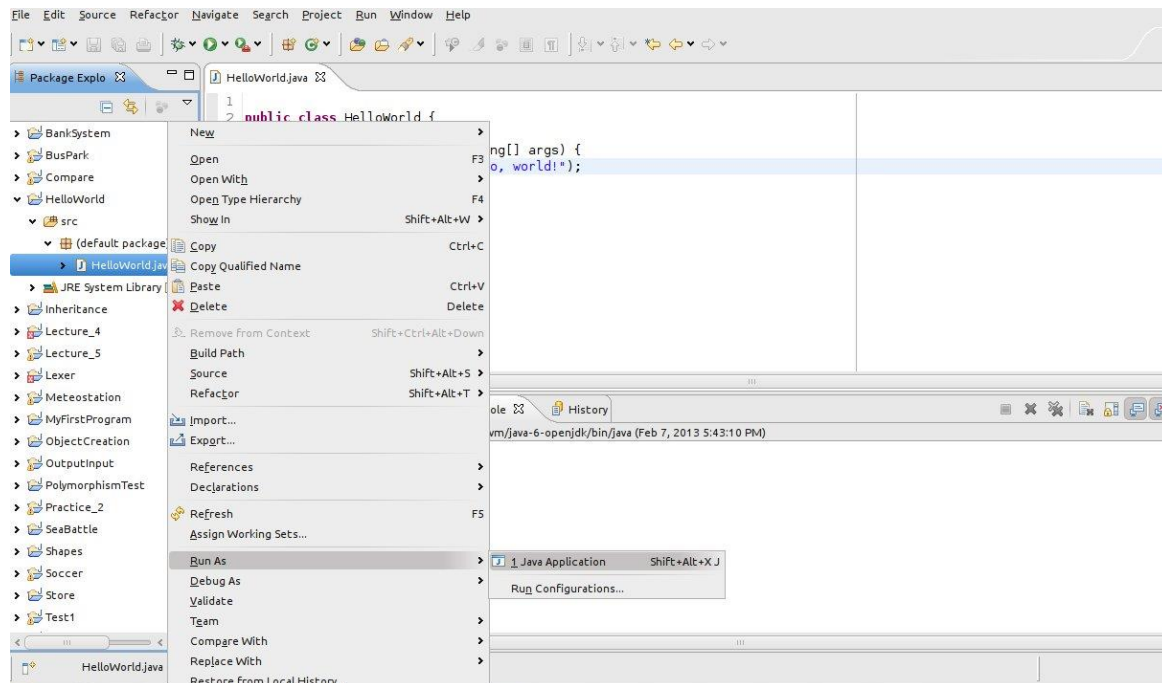


Рис. 7. Запуск приложения на языке Java в IDE Eclipse

Разберем программу более подробно.

**class HelloWorld {**

Эта строка объявляет класс по имени **HelloWorld**. При создании класса используется ключевое слово **class** вместе с именем класса/именем файла.

**Обратите внимание:** принято, чтобы имя класса начиналось с заглавной буквы.

Ключевое слово **class** используется для объявления нового класса. **HelloWorld** – идентификатор, отображающий название класса. Полное описание класса делается в пределах открытой и закрытой изогнутых фигурных скобок. Фигурные скобки указывают компилятору, где начинается и заканчивается описание класса. Открытие и закрытие изогнутой скобки формируют блок этого класса.

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

Ключевое слово **main()** – основной метод. Это – строка, с которой начинается выполнение программы. Все приложения Java должны иметь один метод **main()**. Давайте расшифруем каждое слово в коде.

Ключевое слово **public** – это спецификатор доступа. Спецификаторы доступа будут рассмотрены позже. Когда члену класса предшествует **public**, то к этому члену возможен доступ из кода, внешнего по отношению к классу, в котором описан данный метод. В данном случае *main*-метод объявлен как **public** так, чтобы JVM мог обратиться к этому методу.

Ключевое слово **static** позволяет методу *main()* вызываться без потребности создавать образец класса. К объекту класса нельзя обратиться, не создав это. Но в этом случае есть копия этого метода, доступного в памяти после того, как класс расположен, даже если не был создан образец этого класса. Это важно, потому что JVM вызывает этот метод в первую очередь. Следовательно, этот метод должен быть как **static** и не должен зависеть от экземпляров любого создаваемого класса.

Ключевое слово **void** говорит компилятору, что метод не возвращает никакого значения.

*main ()* – метод, который исполняет специфическую задачу. Это место, с которого начинается выполнение всех приложений Java. Класс, который не имеет основного метода, может быть успешно откомпилирован, но не может быть выполнен, поскольку он не имеет отправной точки выполнения, которой является *main()*-метод.

*String args []* – один из параметров, который передаётся основному методу. Любая информация, которую мы передаём методу, получена переменными, которые упомянуты в пределах круглой скобки метода. Эти переменные – параметры этого метода. Даже если мы не должны передавать никакой информации методу, название метода должно сопровождаться пустыми круглыми скобками, *args []* (переменная) – массив типа *String*. Параметры, которые передают в командной строке, сохранены в этом массиве.

Открытие и закрытие изогнутой скобки для **main** метода составляют блок метода. Функции, которые будут выполнены от основного метода, должны быть определены в этом блоке.

**System.out.println(«Hello, World!») ;**

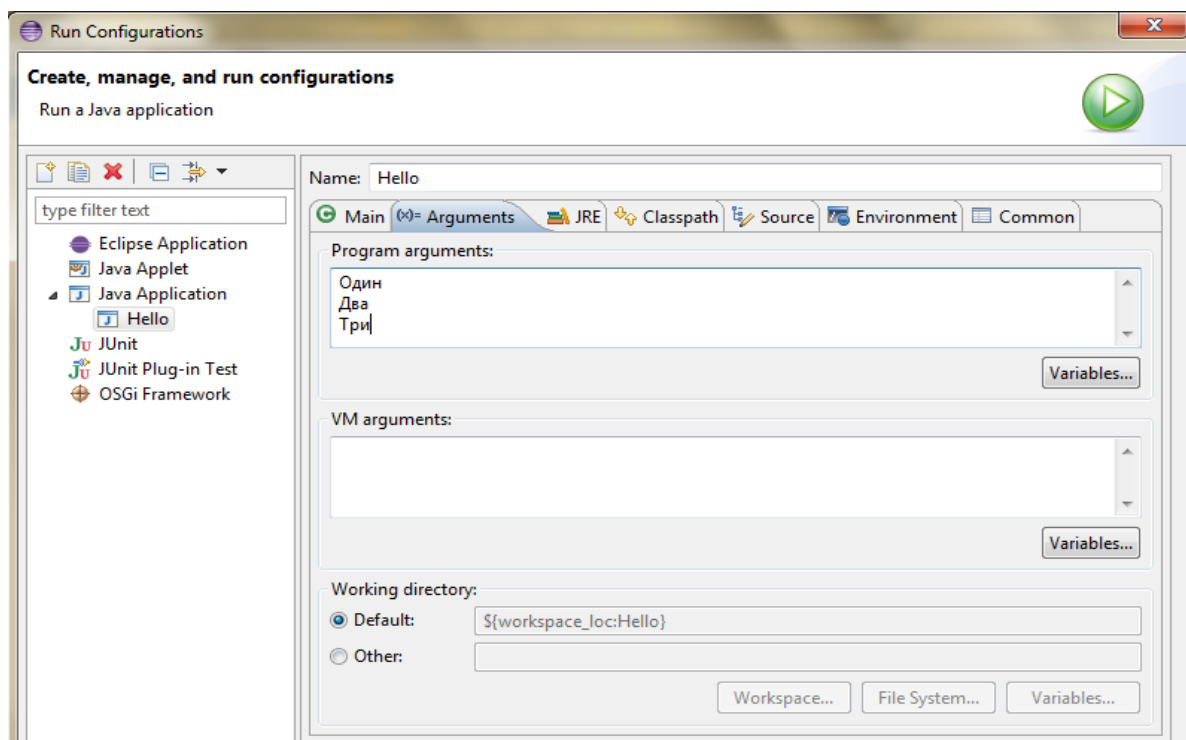
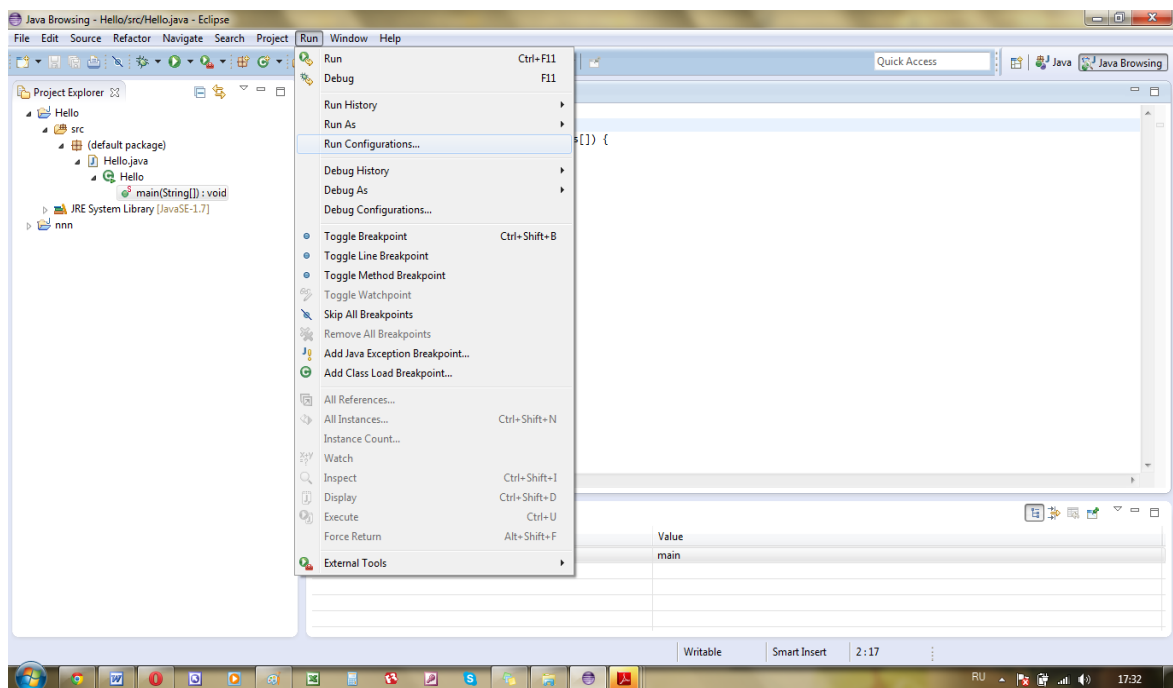
Эта запись отображает строку на экране. Вывод строки осуществляется с помощью метода *println ()*. *println ()* отображает только строку, которая передаётся со справкой *System.out*.

**System** – класс, который является предопределённым и обеспечивает доступ к системе.

**out** – выходной поток и связан с консолью.

Чтобы задать параметры функции *main* можно воспользоваться диалоговым окном **Run/Run Configurations** и перейти на вкладку **Arguments**.





Затем, к параметрам можно обратиться по его порядковому номеру, например, так

```
System.out.println("Привет "+args[0]);
```

Все инструкции в Java заканчиваются точкой с запятой (;).

### Арифметические операторы в Java

Для стандартных арифметических операций, таких как *сложение*, *вычитание*, *умножение*, *деление* в Java используются традиционные символы:

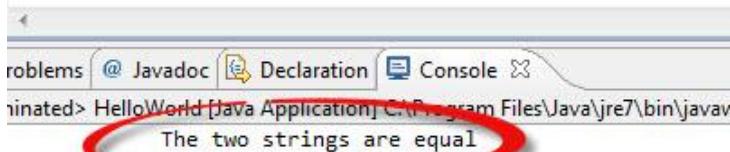
```
int a, b;  
int sum = a + b;  
int sub = a - b;  
int mult = a * b;  
int div = a / b;
```

Операторы == и != тестируют соответственно равенство или неравенство. Для примитивных типов концепция равенства или неравенства достаточно тривиальна. Для значений объектного типа, сравниваемая величина – это ссылка на объект, то есть адрес памяти и для сравнения значений объектных типов используется метод equals() :

"login".equals("login") – true, "login".equals("admin") – false

Например, сравнение двух строк в языке Java на равенство выполняется так:

```
public class HelloWorld {  
    public static void main(String [] args){  
        String myString1 = new String("abc");  
        String myString2 = new String("abc");  
        if(myString1.equals(myString2)){  
            System.out.println("\t\tThe two strings are equal");  
        }  
    }  
}
```



The screenshot shows a Java IDE with a console window. The console output is "The two strings are equal", which is circled in red. The IDE interface includes tabs for "problems", "Javadoc", "Declaration", and "Console". The console window title is "HelloWorld [Java Application] C:\Program Files\Java\jre7\bin\javaw".

### Java: Преобразование типов данных

Конвертация строки в числовой формат на языке Java производится так:

```
byte b = Byte.parseByte("123");  
short s = Short.parseShort("234");  
int i = Integer.parseInt("234");  
long l = Long.parseLong("234");  
float f = Float.parseFloat("234.4");  
double d = Double.parseDouble("233.4e10");
```

### Математические функции и константы (класс Math)

Для решения задач нередко требуется использование математических функций. В Java такие функции включены в класс Math. Для того, чтобы использовать методы класса Math, нужно подключить его в начале .java файла с вашим кодом.

```
import static java.lang.Math.*;
```

Часто используемые математические функции:

- ***sqrt(a)*** — извлекает квадратный корень из числа *a*.
- ***pow(a, n)*** — *a* возводится в степень *n*.
- ***sin(a)*, *cos(a)*, *tan(a)*** — тригонометрические функции *sin*, *cos* и *tg* угла *a* указанного в радианах.
- ***asin(n)*, *acos(n)*, *atan(n)*** — обратные тригонометрические функции, возвращают угол в радианах.
- ***exp(a)*** — возвращает значение экспоненты, возведенной в степень *a*.
- ***log(a)*** — возвращает значение натурального логарифма числа *a*.
- ***log10(a)*** — возвращает значение десятичного логарифма числа *a*.
- ***abs(a)*** — возвращает модуль числа *a*.
- ***round(a)*** — округляет вещественное число до ближайшего целого.

#### Константы

- ***PI*** — число «пи», с точностью в 15 десятичных знаков.
- ***E*** — число «е»(основание экспоненциальной функции), с точностью в 15 десятичных знаков.

Примеры

```
System.out.println(sqrt(81)); // выведет 9.0
System.out.println(pow(2,10)); // выведет 1024.0
System.out.println(sin(PI/2)); // выведет 1.0
System.out.println(cos(PI)); // выведет -1.0
System.out.println(acos(-1)); // выведет 3.141592653589793
System.out.println(round(E)); // выведет 3
System.out.println(abs(-6.7)); // выведет 6.7
```

### 1.3. Самостоятельное задание

1 Реализовать программу, получающую на вход в качестве аргумента *имя человека* и выводящую “*Hello*  + *имя*”, в противном случае, если параметр не передавался, “*Hello world*”.

2 Написать программу, получающую на вход в качестве аргумента *несколько параметров*. В программе вывести “*Вы ввели*” + *N (количество параметров)* + “*параметров*”. Если параметры не передавались, вывести “*Вы не передавали параметров*”.

3 Передавать в качестве параметров *два целочисленных числа*. Вывести на экран как сами значения? так и их сумму (“*3 + 2 = 5*”). Если количество параметров не равно 2, вывести сообщение “*Неверное количество параметров*”.

4 Ввести в качестве параметров *имя пользователя и пароль*. Проверить в методе *main()* соответствие введенных значений заранее определенным значениям. В случае полного соответствия вывести сообщение “*Вас узнали. Добро пожаловать*”, в противном случае вывести сообщение “*Логин и пароль не распознаны. Доступ запрещен*”.