

## Лабораторная работа 1

**Тема:** Создание и запуск первого Java-приложения. Знакомство со средой разработки IntelliJ IDEA.

**Цель:** Изучение процесса создания программ на языке JAVA.

### Теоретическая часть

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к **недостаткам** концепции виртуальной машины относят снижение производительности.

### Классификация платформ Java

Внутри Java существуют несколько основных семейств технологий:

- Java SE — Java Standard Edition, основное издание Java, содержит компиляторы, API, Java Runtime Environment; подходит для создания пользовательских приложений, в первую очередь — для настольных систем.

- Java EE — Java Enterprise Edition, представляет собой набор спецификаций для создания программного обеспечения уровня предприятия.

- Java ME — Java Micro Edition, создана для использования в устройствах, ограниченных по вычислительной мощности, например, в мобильных телефонах, КПК, встроенных системах;

- JavaFX — технология, являющаяся следующим шагом в эволюции Java как Rich Client Platform; предназначена для создания графических интерфейсов корпоративных приложений и бизнеса.

- Java Card — технология предоставляет безопасную среду для приложений, работающих на смарт-картах и других устройствах с очень ограниченным объёмом памяти и возможностями обработки.

### Основные возможности языка JAVA

- автоматическое управление памятью;
- расширенные возможности обработки исключительных ситуаций;
- богатый набор средств фильтрации ввода-вывода;
- набор стандартных коллекций: массив, список, стек и т. п.;
- наличие простых средств создания сетевых приложений (в том числе с использованием протокола RMI);
- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;
- встроенные в язык средства создания многопоточных приложений, которые потом были портированы на многие языки (например, python);
- унифицированный доступ к базам данных:
  - на уровне отдельных SQL-запросов — на основе JDBC, SQLJ;

- на уровне концепции объектов, обладающих способностью к хранению в базе данных — на основе Java Data Objects (англ.) и Java Persistence API;
- поддержка обобщений (начиная с версии 1.5);
- поддержка лямбд, замыканий, встроенные возможности функционального программирования;
- параллельное выполнение программ.

### ***Пространство имён JAVA***

Идея пространств имён воплощена в Java-пакетах.

Внутри пакета есть два независимых пространства имен: переменные и методы.

Java package (пакет Java) — механизм, позволяющий организовать Java классы в пространстве имен аналогично модулям.

Java пакеты могут содержаться в сжатом виде в JAR файлах. Обычно в пакеты объединяют классы одной и той же категории, либо предоставляющие сходную функциональность.

- Каждый пакет предоставляет уникальное пространство имен для своего содержимого.
- Допустимы вложенные пакеты.

Классы, определенные без явно заданных модификаторов доступа (public, protected, private), видимы только внутри пакета.

***private*** — доступ только внутри класса (наиболее рекомендуемый)  
(без модификатора) — только внутри пакета (по умолчанию)

***protected*** — межпакетный доступ только для подклассов

***public*** — межпакетный доступ (наименее рекомендуемый)

### ***Основные идеи JAVA***

#### ***Примитивные типы***

В языке Java только 8 примитивных (скалярных, простых) типов: boolean, byte, char, short, int, long, float, double. Существует также вспомогательный девятый примитивный тип — void, однако переменные и поля такого типа не могут быть объявлены в коде, а сам тип используется только для описания соответствующего ему класса, для использования при рефлексии (процесс, во время которого программа может отслеживать и модифицировать собственную структуру и поведение во время выполнения).

Длины и диапазоны значений примитивных типов определяются стандартом, а не реализацией.

#### ***Объектные переменные, объекты, ссылки и указатели***

В языке Java имеются только динамически создаваемые объекты. Причём переменные объектного типа и объекты в Java — совершенно разные сущности. Переменные объектного типа являются ссылками, то есть неявными указателями на динамически создаваемые объекты. Это подчёркивается синтаксисом описания переменных. Так, в Java нельзя писать:

```
double a[10][20];
```

```
Foo b(30);
```

а нужно:

```
double[][] a = new double[10][20];
```

```
Foo b = new Foo(30);
```

При присваиваниях, передаче в подпрограммы и сравнениях объектные переменные ведут себя как указатели, то есть присваиваются, копируются и сравниваются адреса объектов. А при доступе с помощью объектной переменной к полям данных или методам объекта не требуется никаких специальных операций разыменовывания — этот доступ осуществляется так, как если бы объектная переменная была самим объектом.

Объектными являются переменные любого типа, кроме примитивного. Явных указателей в Java нет. В отличие от указателей C, C++ и других языков программирования, ссылки в Java в высокой степени безопасны благодаря жёстким ограничениям на их использование, в частности:

- Нельзя преобразовывать объект типа `int` или любого другого примитивного типа в указатель или ссылку и наоборот.

- Над ссылками запрещено выполнять операции `++`, `--`, `+`, `-` или любые другие арифметические операции.

Преобразование типов между ссылками жёстко регламентировано. За исключением ссылок на массивы, разрешено преобразовывать ссылки только между наследуемым типом и его наследником, причём преобразование наследуемого типа в наследующий должно быть явно задано и во время выполнения производится проверка его осмысленности. Преобразования ссылок на массивы разрешены лишь тогда, когда разрешены преобразования их базовых типов, а также нет конфликтов размерности.

В Java нет операций взятия адреса или взятия объекта по адресу. Благодаря таким специально введенным ограничениям в Java невозможно прямое манипулирование памятью на уровне физических адресов (хотя определено значение ссылки, не указывающей ни на что: `null`).

Если нужен указатель на примитивный тип, используются классы-обёртки примитивных типов: `Boolean`, `Byte`, `Character`, `Short`, `Integer`, `Long`, `Float`, `Double`.

### Дублирование ссылок и клонирование

Из-за того, что объектные переменные являются ссылочными, при присваивании не происходит копирования объекта. Так, если написать

```
Foo foo, bar;
```

```
...
```

```
bar = foo;
```

то произойдет копирование адреса из переменной `foo` в переменную `bar`. То есть `foo` и `bar` будут указывать на одну и ту же область памяти, то есть на один и тот же объект; попытка изменить поля объекта, на который ссылается переменная `foo`, будет менять объект, с которым связана переменная `bar`, и наоборот. Если же необходимо получить именно ещё одну копию исходного объекта, пользуются или методом (функцией-членом, в терминологии C++) `clone()`, создающим копию объекта, или (реже) копирующим конструктором (конструкторы в Java не могут быть виртуальными, поэтому экземпляр класса-потомка будет неправильно скопирован конструктором класса-предка; метод клонирования вызывает нужный конструктор и тем самым позволяет обойти это ограничение).

### Инициализация переменных

Все переменные или требуют явного определения, или автоматически заполняются нулями (0, `null`, массивом нулей). Таким образом, исчезают ошибки, связанные со случайным использованием неинициализированной памяти, характерные для низкоуровневых языков вроде C.

### Сборка мусора

В языке Java невозможно явное удаление объекта из памяти — вместо этого реализована **Сборка мусора**. Традиционным приёмом, дающим сборщику мусора «намёк» на освобождение памяти, является присваивание переменной пустого значения `null`. Это, однако, не значит, что объект, заменённый значением `null`, будет непременно и немедленно удалён, но есть гарантия, что этот объект будет удалён именно в будущем. Данный приём всего лишь устраняет ссылку на объект, то есть отвязывает указатель от объекта в памяти. При этом следует учитывать, что объект не будет удален сборщиком мусора, пока на него указывает хотя бы одна ссылка из используемых переменных или

объектов. Существуют также методы для инициации принудительной сборки мусора, но не гарантируется, что они будут вызваны исполняющей средой, и их не рекомендуется использовать для обычной работы.

### Классы и функции

Java не является процедурным языком: любая функция может существовать только внутри класса. Это подчёркивает терминология языка Java, где нет понятий «функция» или «функция-член» (англ. member function), а только метод. В методы превратились и стандартные функции. Например, в Java нет функции `sin()`, а есть метод `Math.sin()` класса `Math` (содержащего, кроме `sin()`, методы `cos()`, `exp()`, `sqrt()`, `abs()` и многие другие). Конструкторы в Java не считаются методами. Деструкторов в Java не существует, а метод `finalize()` ни в коем случае нельзя считать аналогом деструктора.

### Конструкторы

Конструктор — это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути, конструктор нужен для автоматической инициализации переменных.

Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.

```
private int Cat(); // так выглядит метод по имени Cat
Cat(); // так выглядит конструктор класса Cat
```

В отличие от метода, конструктор никогда ничего не возвращает.

Конструктор определяет действия, выполняемые при создании объекта класса, и является важной частью класса. Как правило, программисты стараются явно указать конструктор. Если явного конструктора нет, то Java автоматически создаст его для использования по умолчанию.

Например, ниже создается класс `Box` с конструктором, который просто установит начальные значения для объекта класса.

```
class Box {
    int width; // ширина коробки
    int height; // высота коробки
    int depth; // глубина коробки

    // Конструктор
    Box(int a, int b) {
        width = a;
        height = b;
        depth = 10;
    }

    // вычисляем объём коробки
    int getVolume() {
        return width * height * depth;
    }
}
```

Даже если конструктор специально НЕ определён, виртуальная машина Java обязательно его создаст (пустым).

### Статические методы и поля

В Java (как и в C++) используются статические методы (англ. static method — в теории программирования их также называют методами класса), которые задаются при помощи ключевого слова `static`. Статические поля (переменные класса) имеют тот же

смысл, что и в C++: каждое такое поле является собственностью класса, поэтому для доступа к статическим полям не требуется создавать экземпляры соответствующего класса.

Например, математические функции, реализованные в классе `Math`, представляют собой как раз статические методы данного класса. Поэтому можно писать

```
double x = Math.sin(1);
```

вместо

```
Math m = new Math();
```

```
double x = m.sin(1);
```

Поскольку статические методы существуют независимо от объектов (экземпляров класса), они не имеют доступа к обычным (нестатическим) полям и методам данного класса.

Благодаря возможности статического импорта, возможно также вызывать статические функции и константы без указания класса, чтобы вместо такого кода

```
double x = Math.sin(Math.tan(Math.sqrt(y)) + Math.floor(24.5)) +  
Math.cos(42 * Math.PI);
```

писать такой

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

```
...
```

```
double x = sin(tan(sqrt(y)) + floor(24.5)) + cos(42 * PI);
```

### Завершённость

Ключевое слово `final` (финальный) имеет разные значения при описании поля, метода или класса.

1. Финальное поле класса инициализируется при описании или в конструкторе класса (а статическое поле — в статическом блоке инициализации). Впоследствии его значение не может быть изменено. Если статическое поле класса или переменная проинициализированы константным выражением, они рассматриваются компилятором как именованная константа; в таком случае их значение может быть использовано в операторах `switch` (для констант типа `int`), а также для условной компиляции (для констант типа `boolean`) при использовании с оператором `if`.

2. Значения локальных переменных, а также параметров метода, помеченных ключевым словом `final`, не могут быть изменены после присвоения. При этом их значения могут использоваться внутри анонимных классов.

3. Метод класса, отмеченный словом `final`, не может быть переопределён при наследовании.

4. Финальный класс не может иметь наследников.

### Абстрактность

В Java методы, не объявленные явно как `static`, `final` или `private`, являются виртуальными в терминологии C++: при вызове метода, по-разному определённого в базовом и наследующем классах, всегда производится проверка времени выполнения.

Абстрактным методом (модификатор `abstract`) в Java называется метод, для которого заданы параметры и тип возвращаемого значения, но не задано тело. Абстрактный метод определяется в классах-наследниках. Аналог абстрактного метода в C++ — чисто виртуальная функция (`pure virtual function`). Для того чтобы в классе можно было описывать абстрактные методы, сам класс тоже должен быть описан как абстрактный. Объекты абстрактного класса создавать нельзя.

### Интерфейсы

Высшей степенью абстрактности в Java является интерфейс (`interface`). Все методы интерфейса абстрактны: описатель `abstract` даже не требуется. Интерфейс в Java не считается классом, хотя, по сути, является полностью абстрактным классом. Класс может

наследовать/расширять (extends) другой класс или реализовывать (implements) интерфейс. Кроме того, интерфейс может наследовать/расширять другой интерфейс.

В Java класс не может наследовать более одного класса, зато может реализовывать несколько интерфейсов. Множественное наследование интерфейсов не запрещено, то есть один интерфейс может наследоваться от нескольких.

Интерфейсы можно использовать в качестве типов параметров методов. Нельзя создавать экземпляры интерфейсов.

## IntelliJ IDEA

**IntelliJ IDEA** – это коммерческая среда разработки приложений. Существует бесплатная версия “Community Edition” с ограниченным функционалом и полная коммерческая версия “Ultimate Edition”. Версия “Community” распространяется на основе лицензии Apache 2.0 и включает инструменты тестирования, средства контроля версий, сборки ПО, поддерживает языки Java, Java ME, Groovy, Scala и Clojure.

Также в ограниченной версии поддерживается разработка программ для системы Android, имеются средства разработки пользовательского интерфейса, редактор XML-кода, регулярных выражений, проверка синтаксиса, импорт и экспорт проектов Eclipse. “IntelliJ IDEA Community Edition” легко интегрируется с системами отслеживания ошибок.

Версия “Ultimate” помимо стандартного набора языков программирования версии “Community”, поддерживает PHP, SQL, Ruby, CSS, Python, HTML, JS. Работа с технологией Java EE и фреймворками Hibernate, Rails, Google Web Toolkit, Spring также присутствуют. Среди средств интеграции Microsoft Team Foundation Server, Rational Clear Case и Perforce.

Чтобы получить представление о том, как IntelliJ IDEA поможет вам в разработке и запуске Java-приложений, предлагается создать и запустить простейший пример «Hello, World» в данной программе. Таким образом, вы сможете узнать об основных IDE функциях без необходимости вдаваться в детали кода. Пошаговая инструкция поможет вам не запутаться в тонкостях запуска и настройки программы.

### *Перед началом работы*

Для создания Java-приложений потребуется Java Development Kit (JDK).

### *Создание проекта*

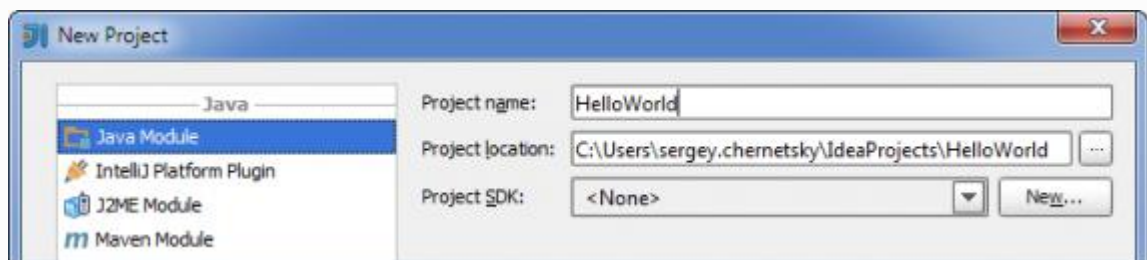
Для начала создадим консольное приложение. Консольное приложение Java представляет собой откомпилированный класс, содержащий точку входа.

Создание любого приложения в IntelliJ IDEA начинается с создания проекта, так что первым нашим шагом будет создание проекта «Hello, World». Этот проект будет содержать модуль Java для нашего приложения Java.

1. Если ни один проект в данный момент не открыт, нажмите кнопку **Create New Project** на экране приветствия. В противном случае, выберите пункт **New Project** в меню **File**. В результате откроется мастер создания проекта.

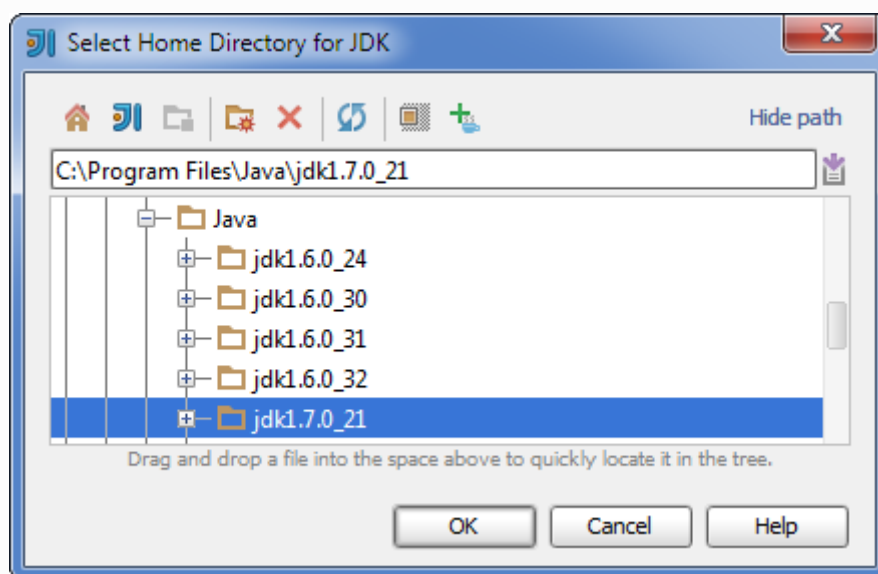
2. В левой панели выберите **Java Module**.

3. В правой части страницы, в поле **Project name**, введите название проекта: *HelloWorld*.

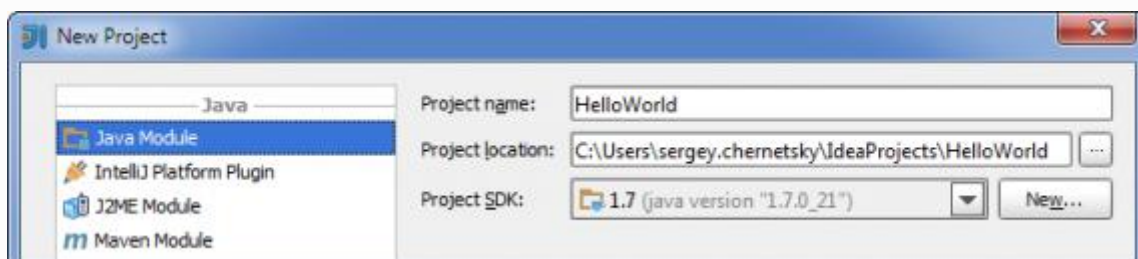


4. Если до этого вы никогда не настраивали JDK в IntelliJ IDEA (в таком случае в поле **Project SDK** стоит **<None>**), необходимо сделать это сейчас. Вместо **<None>**

нажмите **New** и в подменю выберите **JDK**. В окне **Select Home Directory for JDK** выберите директорию, в которую устанавливалось **JDK** и нажмите **OK**.

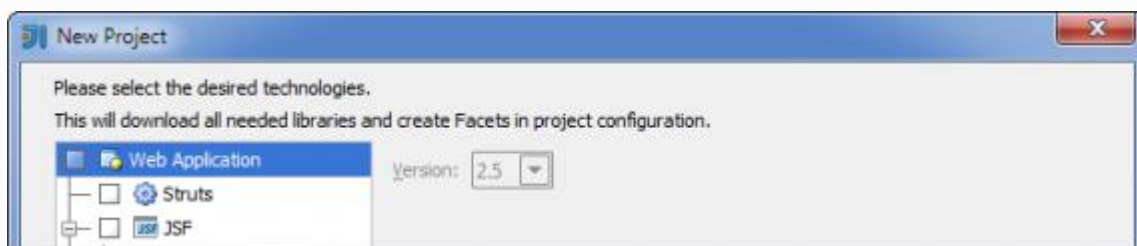


Версия JDK, которую вы выбрали, появится в поле **Project SDK**.



Кликните **Next**. Учтите, что указанная версия JDK будет связана по умолчанию со всеми проектами и Java модулями, которые в дальнейшем будут создаваться.

5. На следующей странице осуществляется выбор мастера для указания дополнительных технологий, которые будут поддерживаться в нашем модуле.

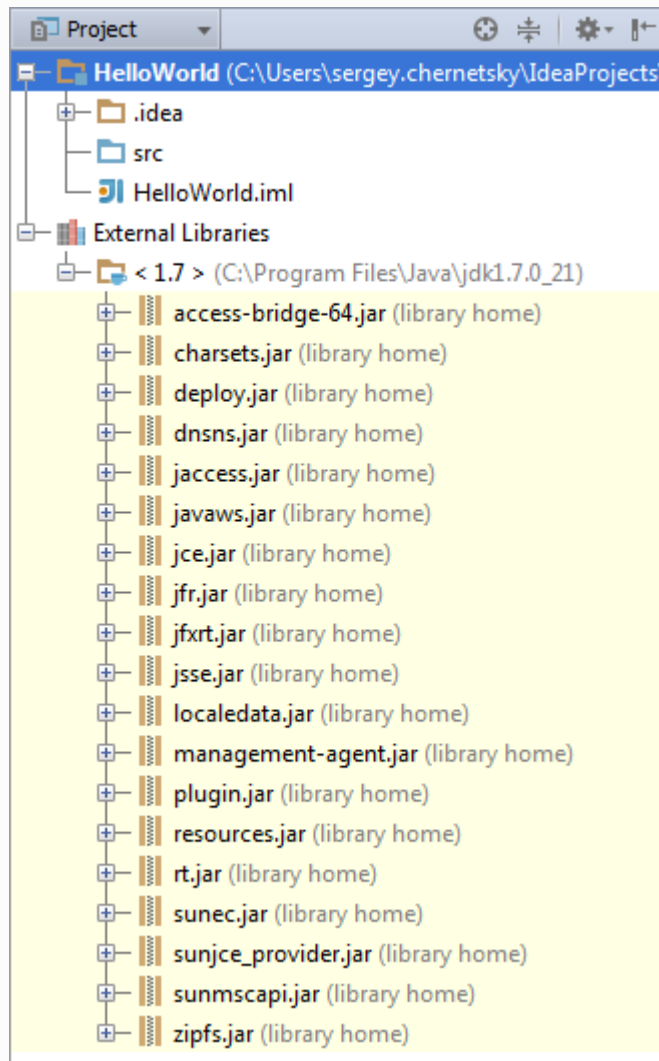


Поскольку наше приложение будет «обычным приложением Java», мы не нуждаемся в любой из этих технологий. Поэтому просто нажмите кнопку **Finish**. Подождите, пока IntelliJ IDEA создает необходимые структуры проекта. Когда этот процесс завершится, вы можете увидеть структуру Вашего нового проекта в окне **Project**.

### Изучение структуры проекта

Давайте пробежимся взглядом по структуре проекта.





В дереве проекта видим две директории верхнего уровня:

- **HelloWorld**. Это узел, содержащий ваш Java модуль. Папки .idea и файлы внутри директории HelloWorld.iml используются для хранения данных конфигурации вашего проекта и модулей соответственно. SRC папки содержат исходный код.
- **External Libraries** (внешние библиотеки). Это категория, которая представляет все «внешние» ресурсы, необходимые для вашего проекта. В настоящее время в этой категории файлы .jar, из выбранного нами JDK.

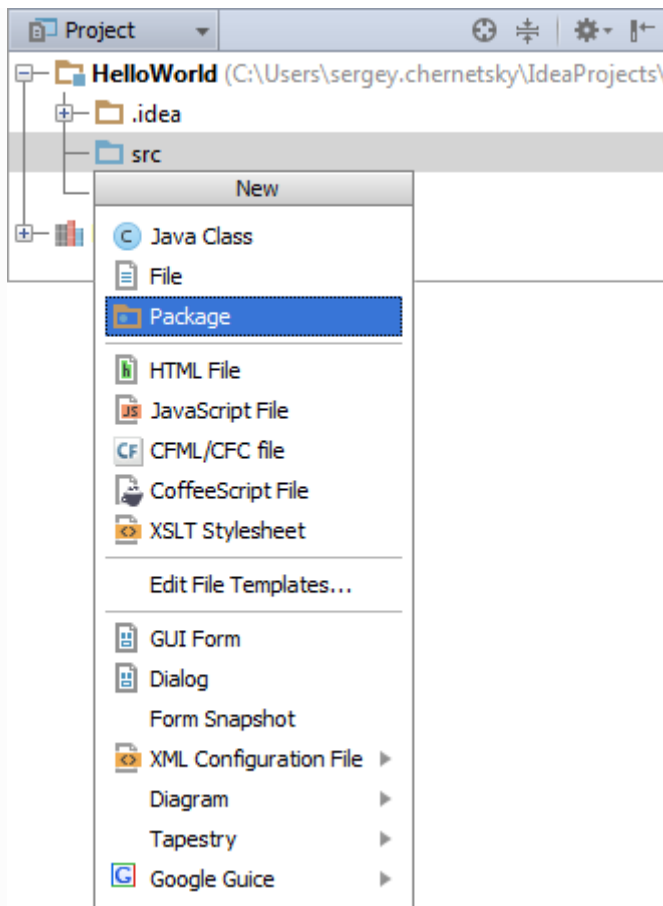
Из всех упомянутых папок в рассматриваемом примере нам понадобится только SRC.

#### *Создание пакета*

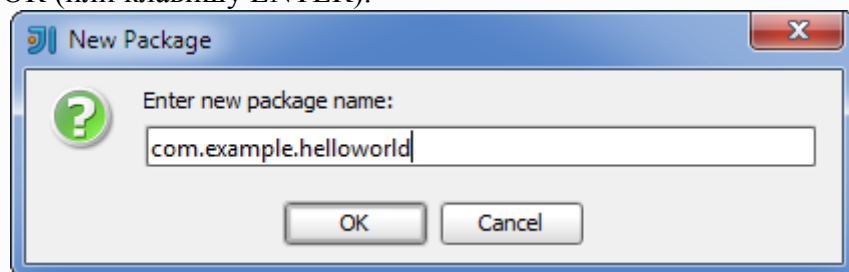
Теперь мы собираемся создать пакет для класса HelloWorld (Мы создадим этот класс чуть позже.) Назовем этот пакет com.example.helloworld.

1. В окне инструментов **Project** выберите папку SRC и нажмите ALT + INSERT. (В качестве альтернативы, вы можете выбрать **File -> New**, или **New** из контекстного меню для папки SRC).

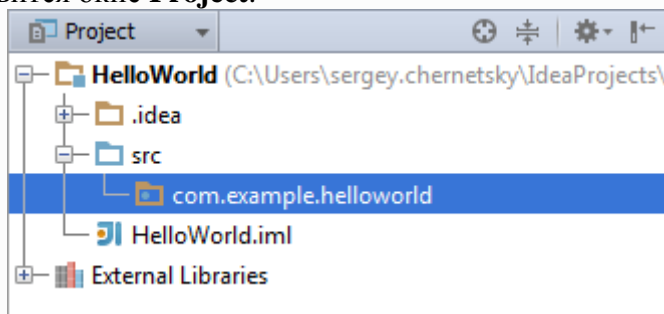
2. В меню **New** выберите **Package**. (можно использовать стрелки вверх и вниз для навигации по меню, ENTER для выбора выделенного элемента)



3. В открывшемся окне **New Package** введите имя пакета (com.example.helloworld). Нажмите кнопку OK (или клавишу ENTER).

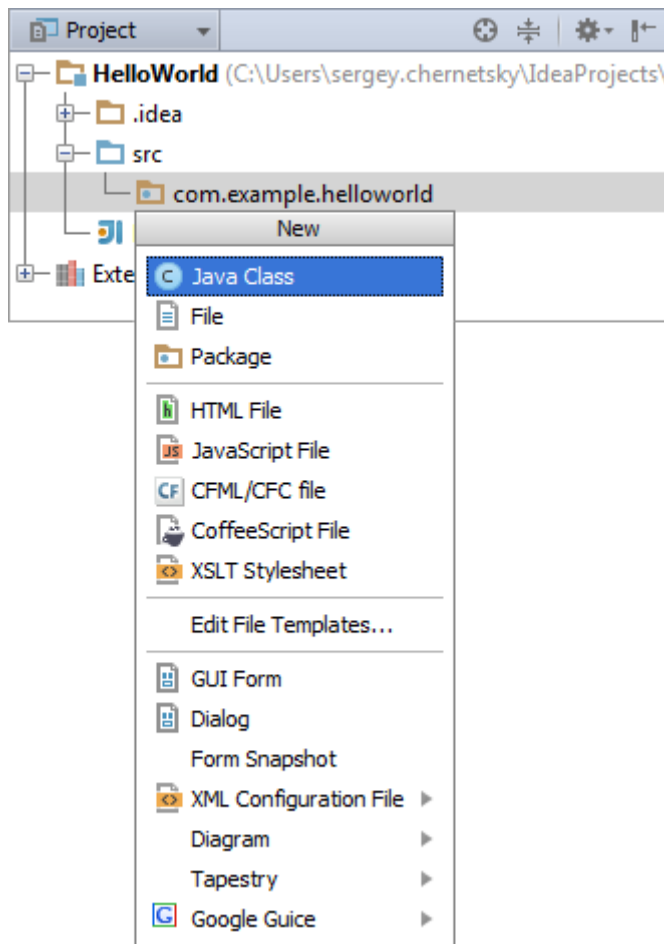


Новый пакет появится окне **Project**.

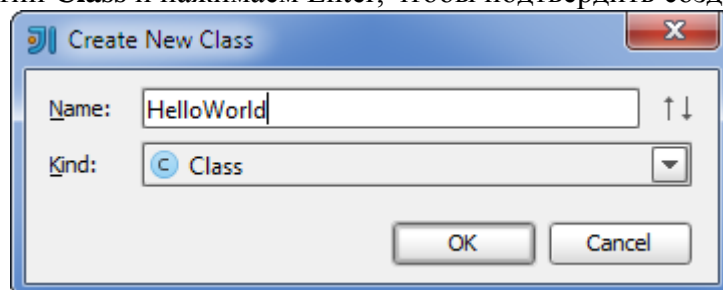


### Создание класса

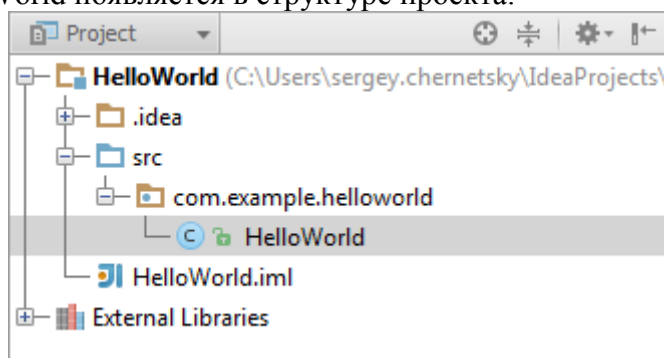
1. Нажмите ALT + INSERT. В окне **New** из списка доступных действий с выделенным пакетом com.example.helloworld выбираем **Java Class** и нажимаем Enter.



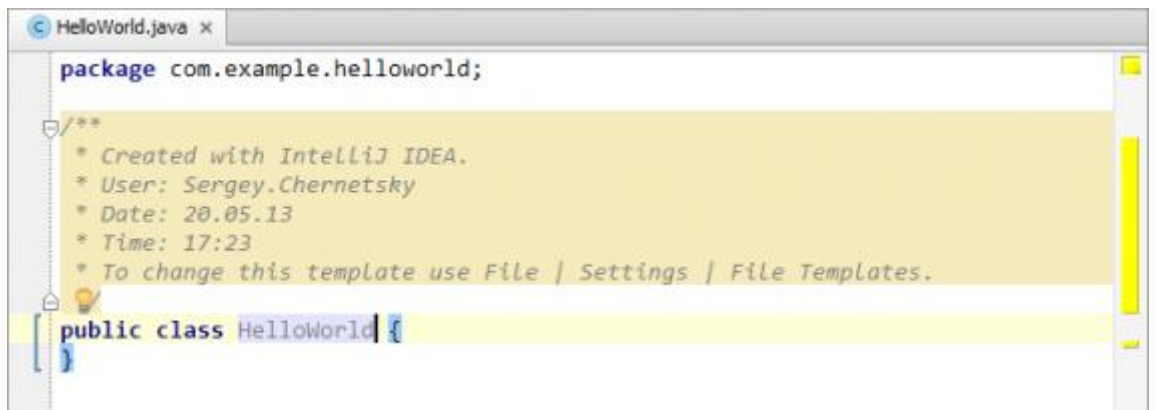
2. В появившемся окне **Create New Class** в поле **Name** вводим имя HelloWorld. В поле **Kind** оставляем тип **Class** и нажимаем Enter, чтобы подтвердить создание класса.



Созданный класс HelloWorld появляется в структуре проекта:

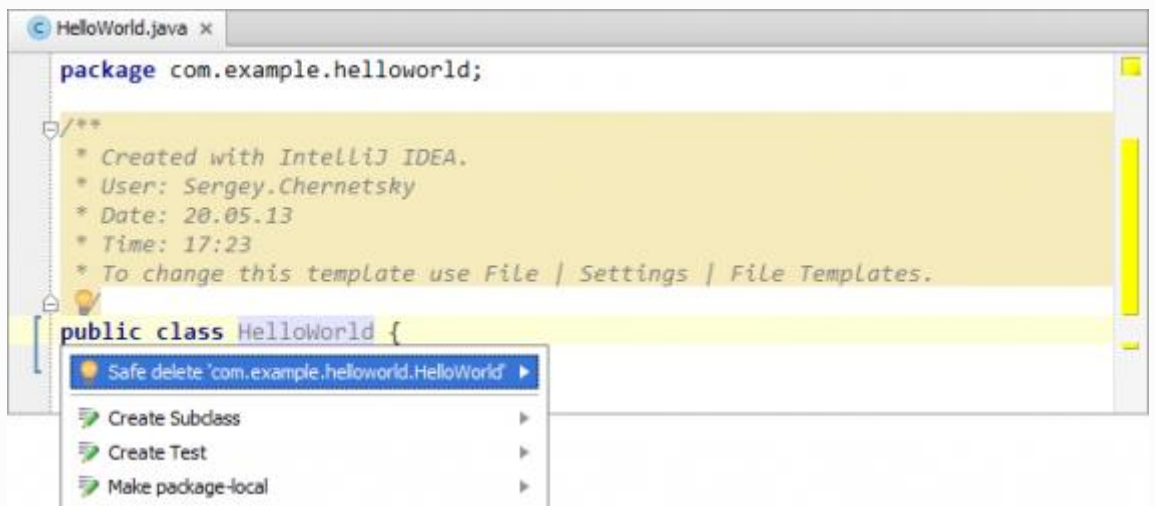


После создания класса соответствующий ему файл HelloWorld.java открывается в редакторе.



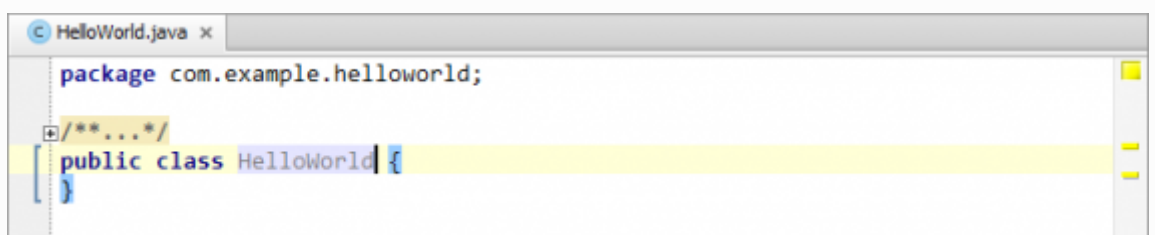
Обратите внимание на оператор пакета в начале файла, а также объявление класса. При создании класса, IntelliJ IDEA использует файл шаблона для класса Java. (IntelliJ IDEA предоставляет ряд предопределенных шаблонов для создания файлов различных типов.

Также обратите внимание на желтую лампочку. Эта лампа указывает, что в IntelliJ IDEA есть предложения для текущего контекста. Нажмите на лампочку или ALT + ENTER, чтобы увидеть список доступных действий.



В данный момент мы не собираемся выполнять действия, предложенные IntelliJ IDEA. Заметим, однако, что эта функция IntelliJ IDEA иногда может быть очень полезной.

Наконец, есть маркеры сворачивания кода рядом с комментариями (☐ и ☐). Нажмите одну из них, чтобы свернуть соответствующий блок, если действительно не хотите видеть эту часть кода в данный момент. (Вы можете также поместить курсор в код блока, а затем нажать сочетание клавиш CTRL + NumPad- или CTRL + NumPad+, чтобы свернуть или развернуть блок. Получить дополнительную информацию по сворачиванию кода можно в разделе [Code Folding](#) в IntelliJ IDEA Help.)



### Написание кода для класса HelloWorld

Код в конечном состоянии будет выглядеть следующим образом:

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

Здесь класс `HelloWorld` используется только для того, чтобы определить метод `main()`, который и является точкой входа и с которого начинается выполнения программы интерпретатором Java. Метод `main()` содержит аргументы-параметры командной строки `String[] args` в виде массива строк и является открытым (`public`) членом класса. Это означает, что метод `main()` виден и доступен любому классу. Ключевое слово `static` объявляет методы и переменные класса, используемые для работы с классом в целом, а не только с объектом класса. **Символы верхнего и нижнего регистров в Java различаются.**

Вывод строки «Hello, World!» в примере осуществляет метод `println()` (ln – переход к новой строке после вывода) свойства `out` класса `System`, который доступен в программе автоматически вместе с пакетом `java.lang`.

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);           //Вывод 7
int j=10; System.out.println("j="+j+(i+1)); // Вывод j=108
```

Все классы являются производными (или подклассами) от существующих классов. В случае класса `HelloWorld` 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 HelloWorld extends Object {
```

### ПРИМЕЧАНИЕ

Так как класс `HelloWorld` 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`).

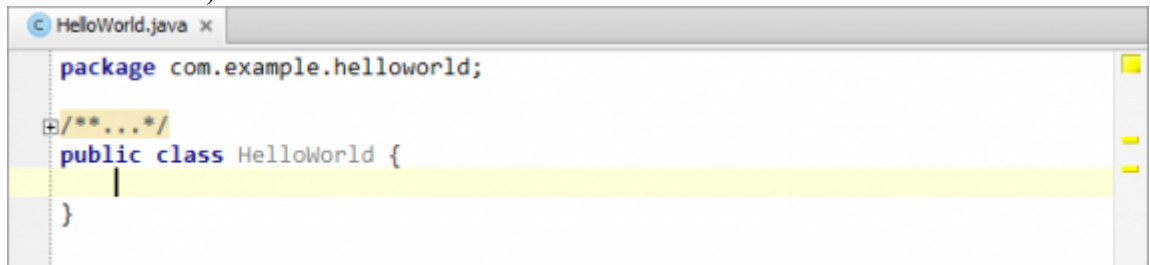
В классе `Hello` объявляется метод `main()` со строковым параметром `args`, который будет содержать аргументы командной строки, передаваемые при запуске приложения:

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

Без функции `main()` нее интерпретатор не сумеет понять, откуда начинать выполнение приложение (метод `main()` является точкой входа приложения). Java-приложения могут запускаться с аргументами командной строки. Хотя необходимо обязательно включать параметр `args` в определение метода `main()`, но использовать аргументы командной строки необязательно.

Объявление пакета и класса уже есть, теперь добавим недостающие пару строк.

Поместите курсор в конец текущей строки, после знака {, и нажмите ENTER, чтобы начать новую строку (На самом деле, можно сделать проще: независимо от позиции курсора, нажатие клавиш SHIFT + ENTER начинает новую строку, сохраняя предыдущие строки без изменений).



```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    |
}
```

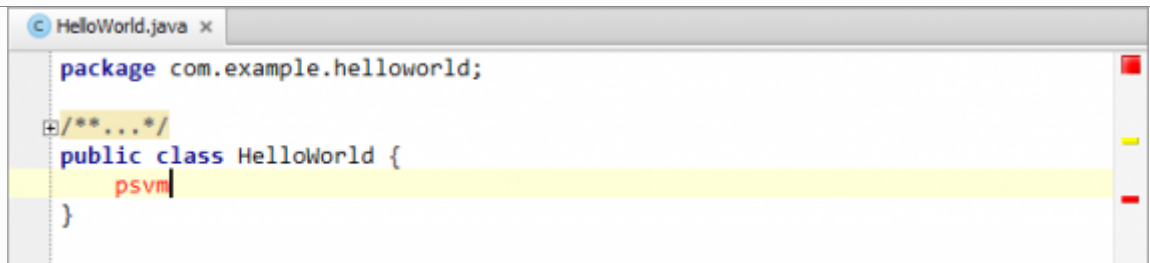
### **Использование активного шаблона для метода Main().**

Строку

```
public static void main(String[] args) {}
```

вполне можно и просто напечатать. Однако есть другой метод. Печатаем:

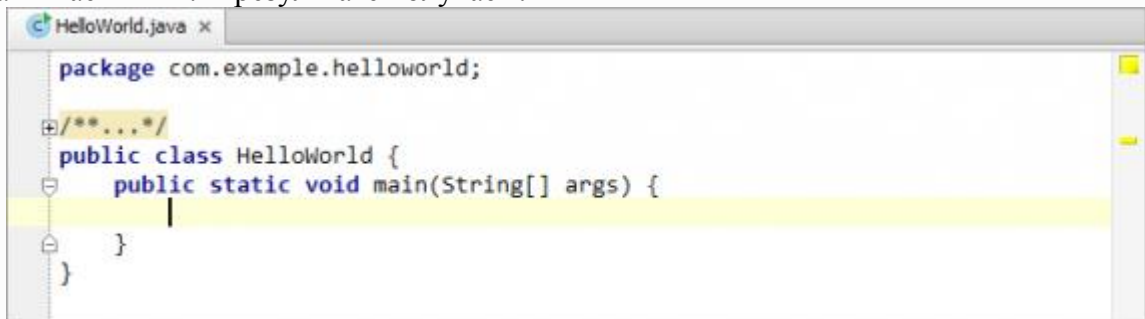
psvm



```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    psvm
}
```

и нажимаем TAB. В результате получаем:



```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    public static void main(String[] args) {
    |
}
}
```

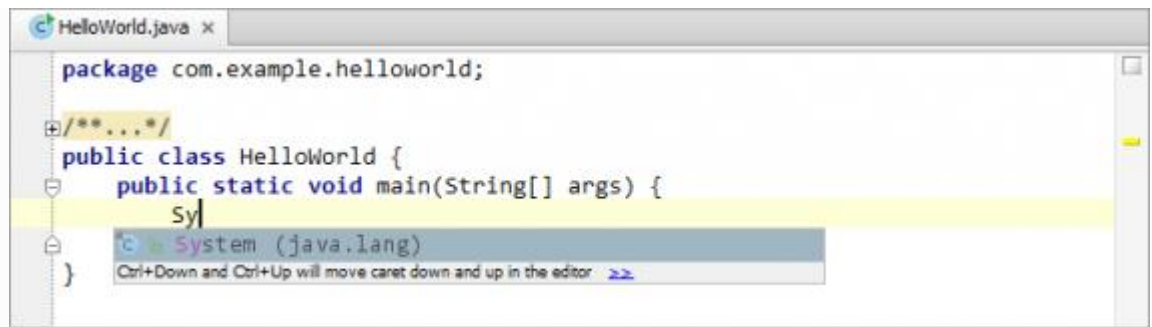
В данном случае мы использовали активный шаблон генерации кода объекта. Активный шаблон имеет аббревиатуру- строку, определяющую шаблон (PSVM = public static void main в этом примере) и клавишу для вставки фрагмента в код (TAB в данном случае).

### **Использование автоматического завершения кода.**

Теперь можно добавить оставшиеся строки кода (System.out.println («Hello, World!»);). Мы сделаем это с помощью операции автоматического завершения кода в IntelliJ IDEA. Печатаем:

Sy

Автоматическое завершение кода предлагает нам варианты:

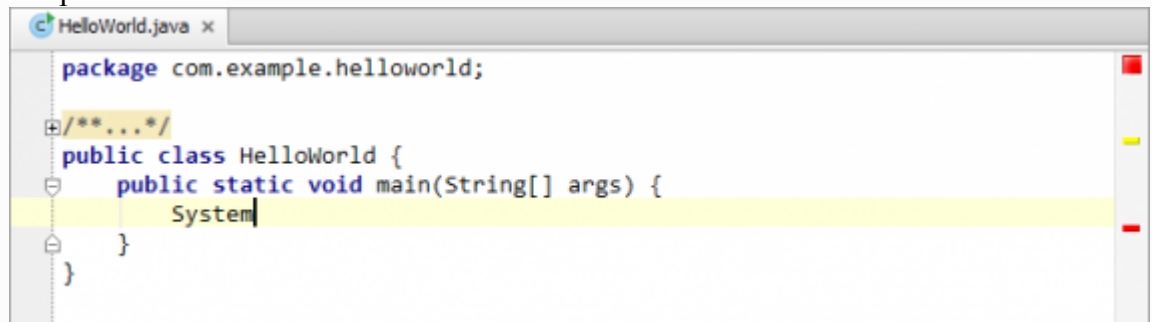


```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    public static void main(String[] args) {
        Sy
    }
}
```

System (java.lang)  
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>

В данном случае вариант только один: **System (java.lang)**. Нажимаем ENTER, чтобы выбрать его.



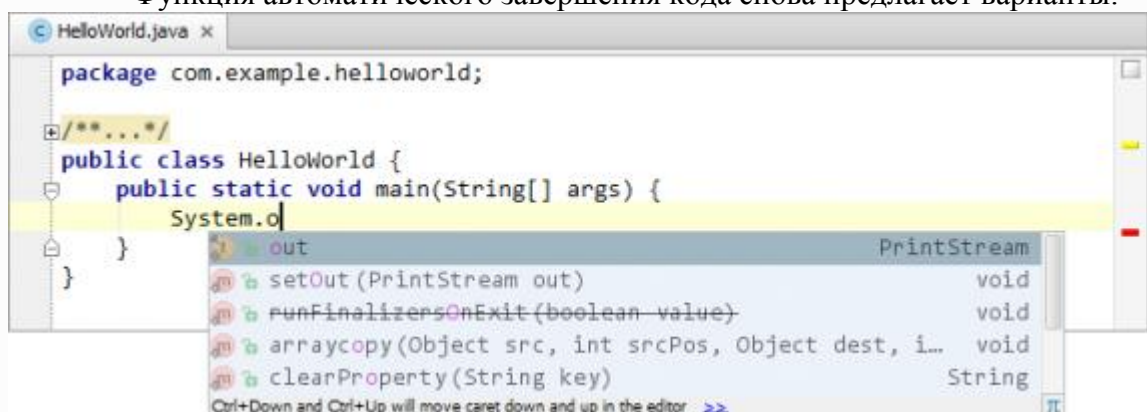
```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    public static void main(String[] args) {
        System
    }
}
```

Печатаем точку и букву «о»:

.o

Функция автоматического завершения кода снова предлагает варианты:

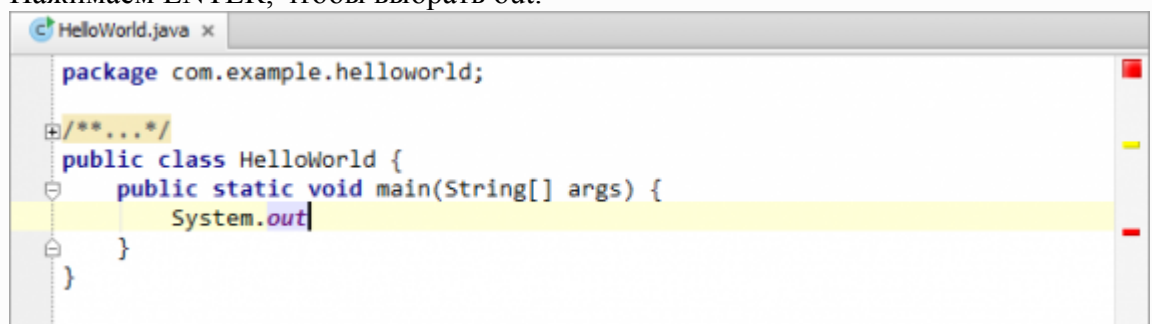


```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    public static void main(String[] args) {
        System.o
    }
}
```

out PrintStream  
setOut(PrintStream out) void  
runFinalizersOnExit(boolean value) void  
arraycopy(Object src, int srcPos, Object dest, i... void  
clearProperty(String key) String  
Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>

Нажимаем ENTER, чтобы выбрать out.



```
package com.example.helloworld;

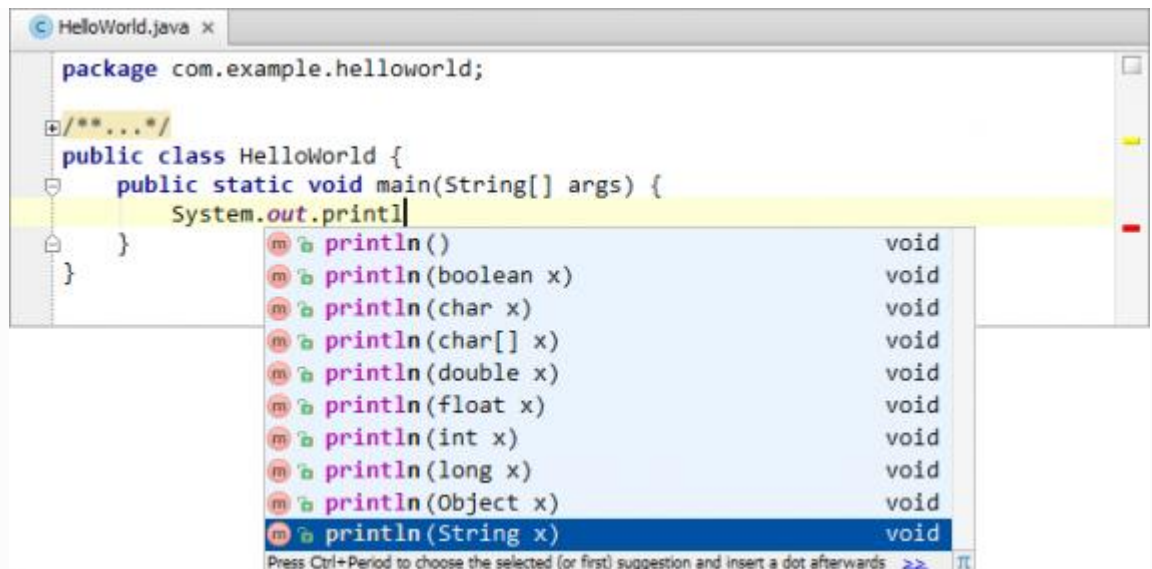
/**...*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out
    }
}
```

Печатаем:

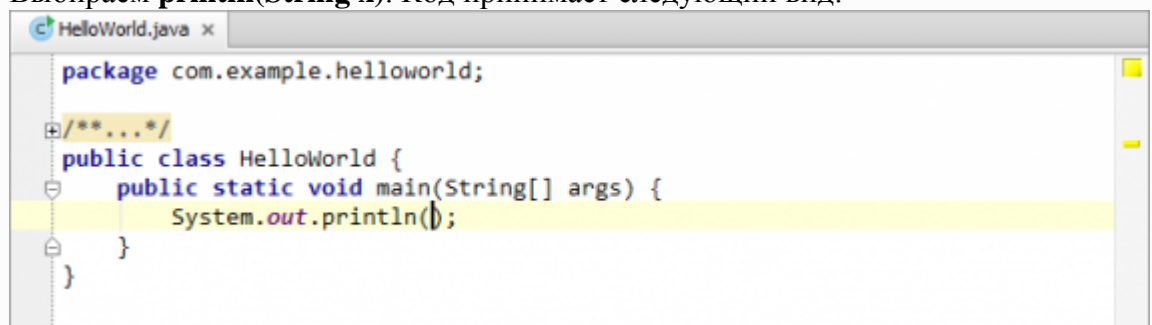
.println

Обратите внимание, как изменяется список вариантов в процессе ввода. Метод, который мы ищем — **Println (String x)**.





Выбираем **println(String x)**. Код принимает следующий вид:

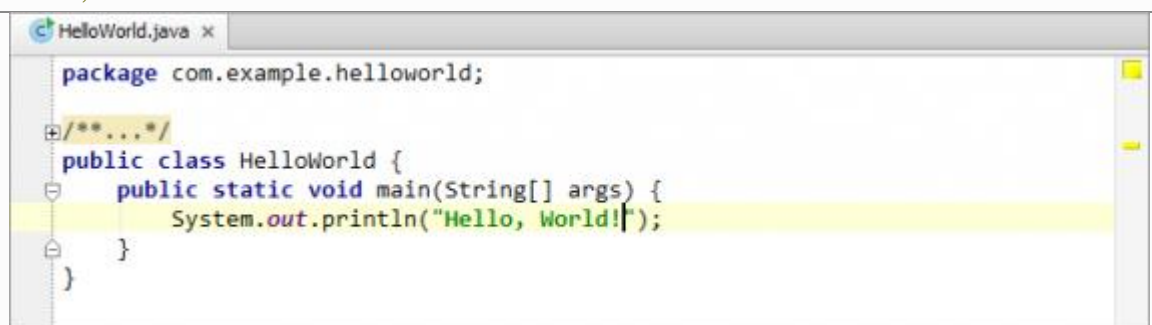


Печатаем кавычки:

"

Вторые кавычки появляются автоматически, а курсор перемещается в место, где должен быть наш текст. Печатаем:

Hello, World!



Этап написания кода завершен.

### **Использование активного шаблона для Println ()**

К слову, мы могли бы выполнить вызов Println () с помощью активного шаблона. Аббревиатура для соответствующего шаблона- **Sout**. а клавиша активации- **ТАВ**. Вы можете попробовать использовать этот шаблон в качестве дополнительного упражнения.

Удалите строку

System.out.println("Hello, World!");

Печатаем

sout

и нажимаем **ТАВ**. Строка



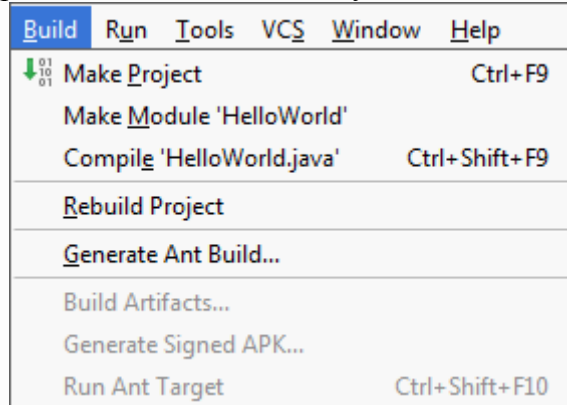
```
System.out.println();
```

добавляется автоматически, и курсор оказывается в скобках. Нам остается напечатать

```
Hello, World!
```

### Построение проекта

Опции построения проекта или его части доступны в меню **Build**.



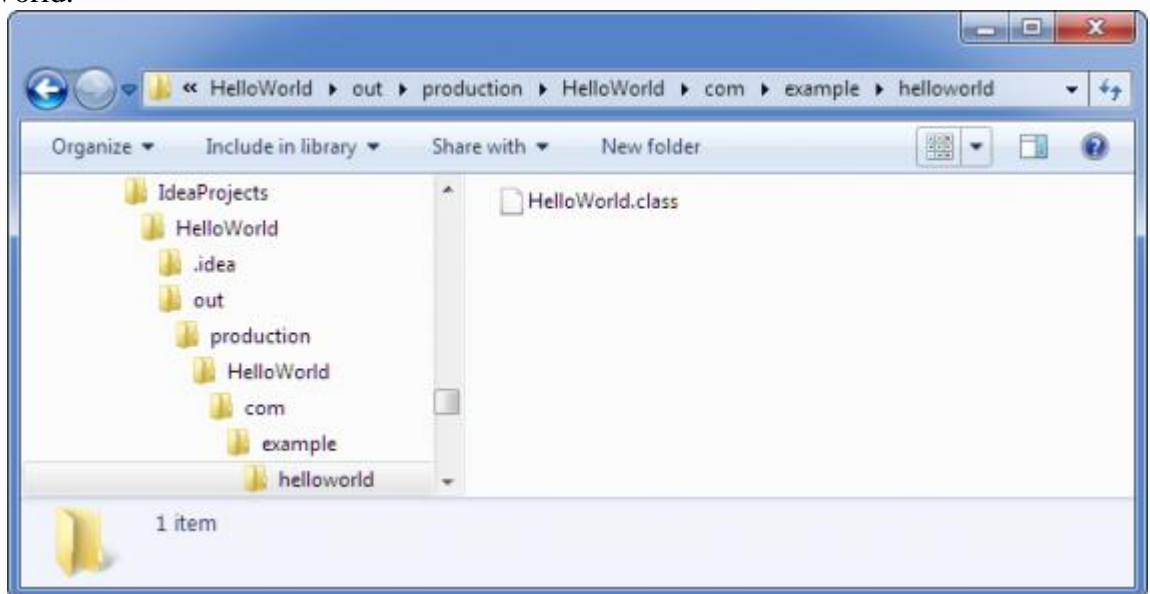
Многие из этих опций доступны также в контекстном меню в окне **Project** и в редакторе для HelloWorld.java. Также есть значок на панели инструментов, которая соответствует команде **Make Project** (📦). Теперь давайте построим проект.

Строительство в данном конкретном случае - просто компиляция исходного файла Java в файл класса. Таким образом, любой из вариантов в меню **Build** (**Make Project**, **Make Module 'HelloWorld'**, или **Compile 'HelloWorld.java'**) могут быть использованы для этой цели.

Клавиатурный эквивалент для команды построения проекта - CTRL + F9. Когда этот процесс компиляции будет завершен, соответствующая информация отображается в строке состояния.

☐ Compilation completed successfully in 15 sec (moments ago)

Теперь, если вы перейдете в папку модуля вывода (по умолчанию это папка \out\production\, в нашем случае, и папка и называются HelloWorld), вы увидите там структуру папок для пакета com.example.helloworld и HelloWorld.class файл в папке HelloWorld.

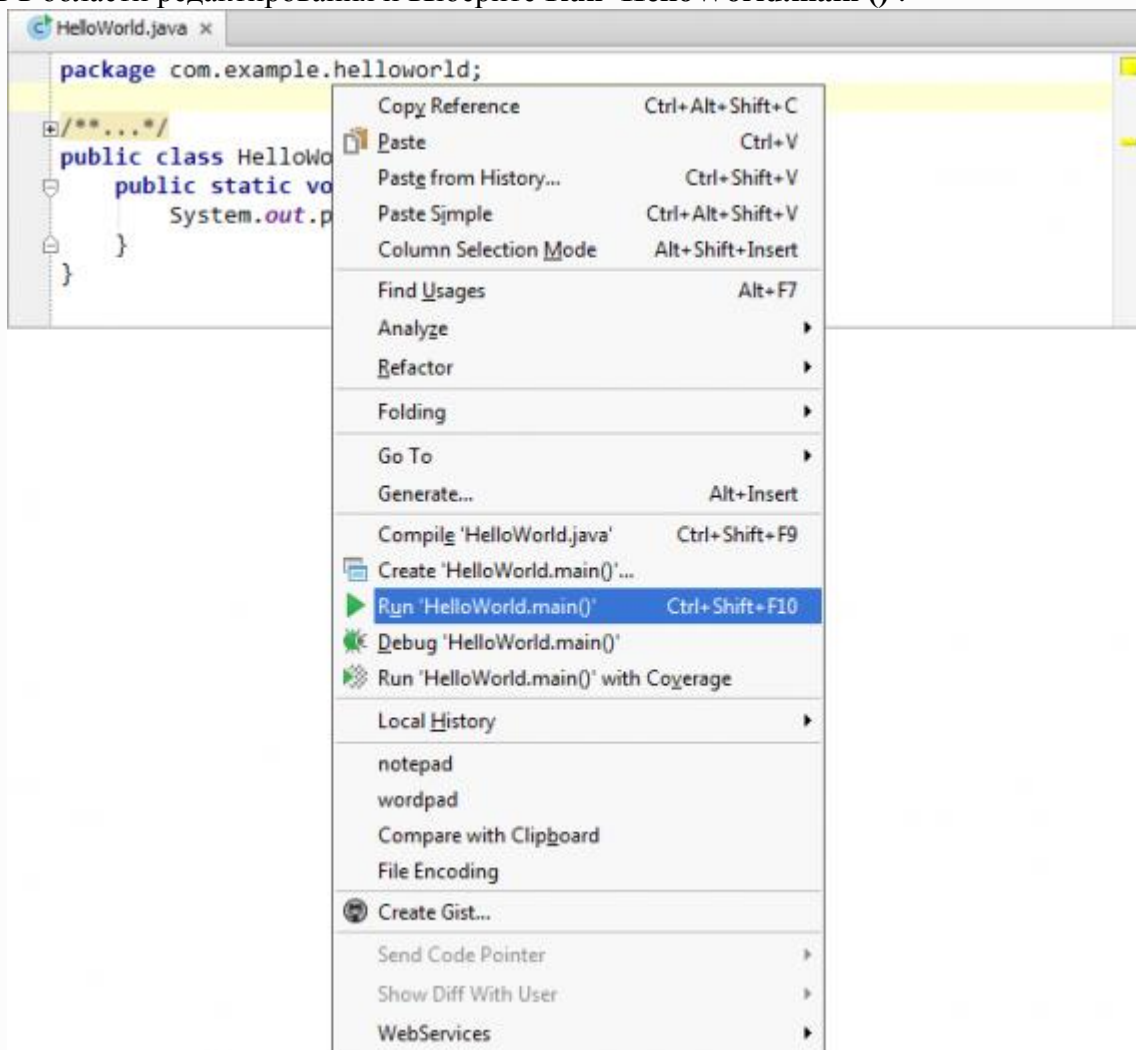


### Запуск приложения

Приложение IntelliJ IDEA выполняется согласно тому, что называется конфигурацией запуска/отладки (Run/Debug). Такая конфигурация, как правило, должна быть создана до запуска приложения.

В случае класса HelloWorld, нет необходимости создавать конфигурацию запуска и отладки заранее. Класс содержит метод **main()**. Такие классы могут быть запущены сразу, прямо из редактора. Для этой цели существует команда **Run '<ClassName>.main()'** в контекстном меню для класса.

Таким образом, чтобы запустить класс, щелкните правой кнопкой мыши где-нибудь в области редактирования и выберите **Run 'HelloWorld.main ()'**.



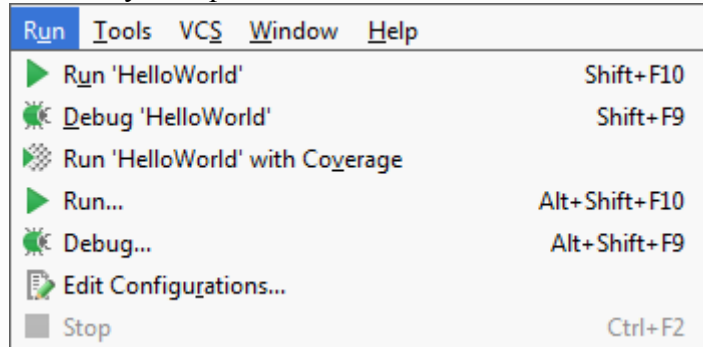
В результате выполнения команды **Run** появляется окно в нижней части экрана. Оно отвечает за отображение всех выходных данных, указанных в конфигурации команды.



Первая строка в окне содержит командную строку IntelliJ IDEA, используемую для запуска класса, включая все опции и аргументы. Последняя строка показывает, что процесс завершился нормально, бесконечных циклов не произошло. И, наконец, вы видите вывод программы Hello, World! между этими двумя строками.

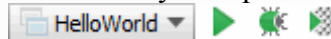
На этом этапе наше упражнение закончено. Однако, есть заключительные замечания, которые стоит сделать, связанные с запуском приложений IntelliJ IDEA:

- Варианты для запуска приложений можно найти в главном меню.

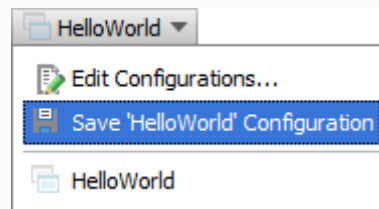


Большинство имен команд в этом меню говорят сами за себя. Опция редактирования конфигурации запуска открывает диалоговое окно для создания и редактирования конфигураций запуска. Также отметим, что сочетания клавиш (см. справа в меню) доступны для большинства команд.

- На главной панели инструментов есть область, содержащая кнопки, связанные с запуском приложений. К ним относятся кнопки выбора конфигурации запуска и отладки (Run/Debug) и значки для запуска приложений в различных режимах.



Выбор конфигурации позволяет выбрать Run/Debug конфигурации, которые вы хотите использовать. Он также позволяет получить доступ к настройке Run/Debug конфигурации (**Edit Configurations**) и выполнения других задач, связанных с работой функций Run/Debug. (В результате запуска класса HelloWorld, Run/Debug конфигурация HelloWorld была сохранена как временная. Теперь вы можете сохранить эту конфигурацию запуска (**Save Configuration «HelloWorld»**), чтобы превратить ее в постоянную.)



- Варианты для запуска приложений и для работы с Run/Debug конфигурациями, в случае необходимости, также присутствуют как команды контекстного меню в окне **Project**.

### ПРИМЕЧАНИЕ

Для создания программ на языке JAVA не обязательно пользоваться средами разработки приложений. Текст программы

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

можно написать в обычном текстовом редакторе, таком как Блокнот и сохранить в файле, имя которого совпадает с именем класса (регистр имеет значение) и с расширением java.

Простейший способ компиляции написанной программы – вызов строчного компилятора:

```
javac HelloWorld.java
```

При успешной компиляции создается файл First.class. Этот файл можно запустить на исполнение из командной строки с помощью интерпретатора Java следующим образом:

```
java HelloWorld
```

Изменим программу, чтобы еще можно было выводить на экран текущую дату и время:

```
package com.example.helloworld;

import java.util.Date; // Подключение библиотеки для работы с классом Date

public class HelloWorld extends Object {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        Date d=new Date();
        System.out.println("Date:"+d.toString());
    }
}
```

Для удобной работы с датой и временем в Java используются классы Calendar и Date. Оба класса находятся в библиотеке java.util. Класс Date хранит время в миллисекундах начиная с 1 января 1970 года. Данный класс имеет конструктор по умолчанию, который возвращает текущее время. Кроме этого можно создать объект Date используя конструктор, который принимает количество миллисекунд начиная с 1 января 1970 года. Для получения этого внутреннего времени используется метод getTime(). Кроме этого уже после создания можно изменить время с помощью setTime(long date).

Для того, чтобы отображать дату и время в удобном для вас формате используется SimpleDateFormat:

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class Test {
    public void test()
    {
        Date d = new Date();
        SimpleDateFormat format1 = new SimpleDateFormat("dd.MM.yyyy
hh:mm");
        SimpleDateFormat format2 = new SimpleDateFormat("День dd Месяц
ММ Год yyyy Время hh:mm");
        System.out.println(format1.format(d)); //25.02.2013 09:03
        System.out.println(format2.format(d)); //День 25 Месяц 02 Год
2013 Время 09:03
    }
}
```

Очевидно, что при создании шаблона для отображения даты dd — означает день, ММ — месяц, уууу — год, hh — часы и mm — минуты. В шаблоне могут присутствовать не все единицы, кроме того, как вы увидели выше в качестве разделителя можно использовать любой текст.

Рассмотрим абстрактный класс Calendar. Он позволяет работать с датой в рамках календаря, т.е он умеет прибавлять день, при этом учитывать високосные годы и прочее. Единственной реализацией его является класс GregorianCalendar, также как и у даты конструктор по умолчанию возвращает календарь на текущий день, но вы можете задать его явно указав все параметры:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class Test {
    public void test()
    {
        Calendar c = new GregorianCalendar();//календарь на текущую дату
```

```

25.11.2013      Calendar c2 = new GregorianCalendar(2013, 11, 25); //календарь на
                c2.add(Calendar.DAY_OF_YEAR, 1); //увеличиваем дату на 1 день
                System.out.println(c2.getTime()); // 26.11.2013
                c2.add(Calendar.DAY_OF_YEAR, -1); //уменьшаем дату на 1 день
            }
        }
    }
}

```

Календарь достаточно мощный класс, который позволяет получать названия месяцев и дней недели, увеличивать или уменьшать различные параметры текущей даты, а также получать их. Для удобства работы с ними вам нужно просто разобраться с типами данных с которыми он работает:

- DAY\_OF\_YEAR — день года (0- 365)
- DAY\_OF\_MONTH — день месяца( какой по счету день в месяце 0 — 31)
- WEEK\_OF\_MONTH — неделя месяца
- WEEK\_OF\_YEAR — неделя в году
- MONTH — номер месяца
- Year — номер года
- Calendar.ERA — эра

Т.е большинство методов принимает на вход `Int field`, где в качестве одного из вариантов вы можете выбрать перечисленные выше значения.

```

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class Test {
    public void test()
    {
        25.11.2013      Calendar c = new GregorianCalendar(2013, 11, 25); //календарь на
                        System.out.println(c.get(Calendar.MONTH)); //11
                        System.out.println(c.get(Calendar.YEAR)); //2013
                        System.out.println(c.get(Calendar.DAY_OF_WEEK_IN_MONTH)); //4
                        System.out.println(c.get(Calendar.DAY_OF_WEEK)); //4
                        System.out.println(c.get(Calendar.DAY_OF_YEAR)); //359
                        System.out.println(c.get(Calendar.DAY_OF_MONTH)); //25
    }
}

```

## Базовые типы данных и арифметические операции с ними

В Java существует 8 базовых типов данных. Для тех, кто уже знаком с каким-либо другим языком программирования эти типы будут наверняка хорошо знакомы, но всё же в Java есть свои особенности.

### Boolean

Для того чтобы создать новый объект этого типа с именем `name`, достаточно в коде написать

```
boolean name;
```

По умолчанию он равен `false`. Для того чтобы придать ему определенное значение, пишут

```
name=false; name =true;
```

Точно также можно задать ему значение сразу во время создания, в таком случае достаточно написать

```
boolean name=true;
```

Допустим у нас имеется три объекта типа `boolean` (назовем их `result`, `a1` и `a2`).

Операция вида

```
result = a1 || a2;
```

Сделает переменную `result` равной результату логического ИЛИ (дизъюнкция) переменных `a1` и `a2`.

```
Result = a1 && a2;
```

Сделает переменную `result` равной результату логического И (Конъюнкция) переменных `a1` и `a2`.

```
Result = !a1;
```

Сделает `result` равным логическому отрицанию переменной `a1`.

Таблица булевых логических операторов

№	Оператор	Результат	№	Оператор	Результат
1	&	логическое И (AND)	7	&=	И (AND) с присваиванием
2		логическое ИЛИ (OR)	8	=	ИЛИ (OR) с присваиванием
3	^	логическое исключающее ИЛИ (XOR)	9	^=	исключающее ИЛИ (XOR) с присваиванием
4		оператор OR быстрой оценки выражений (short circuit OR)	10	==	равно
5	&&	оператор AND быстрой оценки выражений (short circuit AND)	11	!=	не равно
6	!	логическое унарное отрицание (NOT)	12	? :	тернарный оператор if-then-else

Существуют два дополнения к набору логических операторов. Это альтернативные версии операторов AND и OR, служащие для быстрой оценки логических выражений. Если первый операнд оператора OR имеет значение `true`, то независимо от значения второго операнда результатом операции будет величина `true`. Аналогично в случае оператора AND, если первый операнд — `false`, то значение второго операнда на результат не влияет — он всегда будет равен `false`. Если вы используете операторы `&&` и `||` вместо

обычных форм `&` и `|`, то Java не производит оценку правого операнда логического выражения, если ответ ясен из значения левого операнда. Общепринятой практикой является использование операторов `&&` и `||` практически во всех случаях оценки булевых логических выражений. Версии этих операторов `&` и `|` применяются только в битовой арифметике.

### Числа

Числовые типы данных объявляются также как и `boolean`, и их значение по умолчанию это 0 для целочисленных типов, 0.0F для `float` и 0.0D для `double`.

Операции с численными типами данных по большей части одинаковы. Например, пусть имеется 3 числа(назовем их `result`, `x1` и `x2`).

```
result = x1 + x2;
```

Запишет в переменную `Result` сумму `x1` и `x2`. По такому же принципу используются операторы вычитания (`—`) и умножения (`*`). А также деление (`/`) и остаток от деления (`%`).

Это самые основные операции с численными типами. Также стоит упомянуть про сравнение чисел. Пусть имеется `Boolean` переменная `Result` и две переменные (не обязательно `Boolean`) `a1` и `a2`.

#### Операция

```
result = (a1 == a2);
```

Запишет в `result` значение `true` если `a1` равно `a2`, и `false` в противном случае. Знак `!=` это наоборот, отрицание равенства.

Кроме уже перечисленных типов, есть ещё тип `char`

Например:

```
int kod=70;
char sim=(char) kod;
System.out.println(sim);
```

В результате выполнения этого кода в консоль выводится символ с кодом 70, то есть символ "F".

## Виды комментариев

- многострочный (`/*...*/`)
- однострочный (`//...`)

## Консольный ввод и вывод

Для получения данных, введенных пользователем, а также для вывода сообщений нам необходим ряд классов, через которые мы сможем взаимодействовать с консолью. Частично их использование уже рассматривалось в предыдущих темах. Для взаимодействия с консолью нам необходим класс `System`. Этот класс располагается в пакете `java.lang`, который автоматически подключается в программу, поэтому нам не надо дополнительно импортировать данный пакет и класс.

### Вывод на консоль

Для создания потока вывода в класс `System` определен объект `out`. В этом объекте определен метод `println`, который позволяет вывести на консоль некоторое значение с последующим переводом консоли на следующую строку:

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

В метод `println` передается любое значение, как правило, строка, которое надо вывести на консоль. При необходимости можно и не переводить курсор на следующую

строку. В этом случае можно использовать метод `System.out.print()`, который аналогичен `println` за тем исключением, что не осуществляет перевода на следующую строку.

```
System.out.print("Hello world");
```

Но с помощью метода `System.out.print` также можно осуществить перевод каретки на следующую строку. Для этого надо использовать escape-последовательность `\n`:

```
System.out.print("Hello world \n");
```

Если у нас есть два числа, и мы хотим вывести их значения на экран, то мы можем, например, написать так:

```
int x=5;
int y=6;
System.out.println("x="+x +"; y="+y);
```

Но в Java есть также функция для форматированного вывода, унаследованная от языка C: `System.out.printf()`. С ее помощью мы можем переписать предыдущий пример следующим образом:

```
int x=5;
int y=6;
System.out.printf("x=%d; y=%d \n", x, y);
```

В данном случае символы `%d` обозначают спецификатор, вместо которого подставляет один из аргументов. Спецификаторов и соответствующих им аргументов может быть множество. В данном случае у нас только два аргумента, поэтому вместо первого `%d` подставляет значение переменной `x`, а вместо второго - значение переменной `y`. Сама буква `d` означает, что данный спецификатор будет использоваться для вывода целочисленных значений типа `int`.

Кроме спецификатора `%d` мы можем использовать еще ряд спецификаторов для других типов данных:

`%x`: для вывода шестнадцатеричных чисел

`%f`: для вывода чисел с плавающей точкой

`%e`: для вывода чисел в экспоненциальной форме, например, `1.3e+01`

`%c`: для вывода одиночного символа

`%s`: для вывода строковых значений

### **Консольный ввод**

Для получения консольного ввода в классе `System` определен объект `in`. Однако непосредственно через объект `System.in` не очень удобно работать, поэтому, как правило, используют класс `Scanner`, который, в свою очередь использует `System.in`. Например, создадим маленькую программу, которая осуществляет ввод чисел:

```
import java.util.Scanner;
public class FirstApp {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int[] nums = new int[5];
        for(int i=0; i < nums.length; i++){
```



```

        nums[i]=in.nextInt();
    }
    for(int i=0;i < nums.length; i++){
        System.out.print(nums[i]);
    }
    System.out.println();
}
}

```

Так как класс Scanner находится в пакете java.util, то мы вначале его импортируем. Для создания самого объекта Scanner в его конструктор передается объект System.in. После этого мы можем получать вводимые значения. Например, чтобы получить введенное число, используется метод in.nextInt(), который возвращает введенное с клавиатуры целочисленное значение.

В данном случае в цикле вводятся все элементы массива, а с помощью другого цикла все ранее введенные элементы массива выводятся в строчку.

Класс Scanner имеет еще ряд методов, которые позволяют получить введенные пользователем значения:

- next(): считывает введенную строку до первого пробела
- nextLine(): считывает всю введенную строку
- nextInt(): считывает введенное число int
- nextDouble(): считывает введенное число double
- hasNext(): проверяет, было ли введено слово
- hasNextInt(): проверяет, было ли введено число int
- hasNextDouble(): проверяет, было ли введено double

Кроме того, класс Scanner имеет еще ряд методов nextByte/nextShort/nextFloat/nextBoolean, которые по аналогии с nextInt считывают данные определенного типа данных.

Создадим следующую программу для ввода информации о человеке:

```

import java.util.Scanner;
public class FirstApp {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Введите имя: ");
        String name = in.nextLine();
        System.out.print("Введите возраст: ");
        int age = in.nextInt();
        System.out.println("Ваше имя:" + name + "Ваш возраст:"
+ age);
    }
}

```

### Задание на лабораторную работу

1. Реализовать пример из теоретической части в среде IntelliJ IDEA, а также без среды разработки (в командной строке).
2. Написать программу, которая выводит на экран ФИО студента, группу (из строковой переменной), а также (по вариантам):
  - Год и месяц запуска программы.
  - Время запуска программы.
  - Час запуска программы и количество дней до Нового года.
3. Создать приложение, вычисляющее значения переменных по заданным расчетным формулам и наборам исходных данных. На экран вывести значения вводимых исходных данных и результаты вычислений, сопровождая ввод и вывод поясняющими комментариями.

Вариант задания	Расчетные формулы	Значения исходных данных
1	$a = \frac{\sin x + 4}{\cos(y^2)}$ $b = \bar{a} * \frac{x}{y}$	x=10 y=0,5
2	$a = \frac{\cos x + \sin(y^2)}{\cos(y^2)}$ $b = \bar{a} + \frac{x}{y}$	x=40 y=0,5
3	$a = \frac{\cos x + \sin(y^2)}{2}$ $b = a \frac{2}{4} + \frac{4x}{y}$	x=4,5 y=0,7
4	$a = \frac{1^y + \sin(y^2)}{2}$ $b = a \frac{2}{4} + \frac{4x}{y}$	x=4,5 y=0,7
5	$a = y^x + \frac{\sin(y^2)}{2x}$ $b = a \frac{2}{4} + \frac{y}{4}$	x=4,5 y= -0,7
6	$a = y^x + \frac{1}{2y + 3/x}$ $b = a \frac{2}{4} + \cos x$	x= -4,5 y= -0,7
7	$a = y^x + \frac{\bar{4}}{2y + 3/x}$ $b = a \frac{2}{4} - \cos x$	x= 2,5 y= -1,7
8	$a = y^{x-0.1} + \frac{\bar{4}}{2y + 3/x}$	x= 2,5 y= -1,7

	$b = \overline{a \frac{2}{4} - \frac{\cos x}{3y}}$	
<b>9</b>	$a = y^x + \frac{\overline{4}}{2y + 3/x}$ $b = \overline{a \frac{2}{4} - \frac{\cos x}{3y}}$	$x = 1,5$ $y = -1,7$
<b>10</b>	$a = y^x + \frac{1}{2y * 3/x}$ $b = a^x + \frac{\overline{4}}{\overline{2} + 3/x}$	$x = 0,5$ $y = -0,7$
<b>11</b>	$a = \overline{a \frac{2}{4} + \frac{4y^3x}{y}}$ $b = \overline{a \frac{2}{4} - \frac{4x}{y}}$	$x = 3,5$ $y = 0,7$
<b>12</b>	$a = \frac{\sin x + 4}{\overline{\cos(y^2)}}$ $b = \overline{a \frac{2}{4} - \frac{4x}{y}}$	$x = 0,55$ $y = 0,47$

4. Создать приложение для решения задачи.

4.1 Определить плату за электроэнергию, если известны: старое и новое показания счетчика, стоимость одного квт/часа, количество просроченных дней, размер пени за один день просрочки.

4.2 Длина отрезка задана в дюймах (1 in = 2,54 см). Перевести значение длины в метрическую систему, то есть выразить ее в метрах, сантиметрах, миллиметрах.