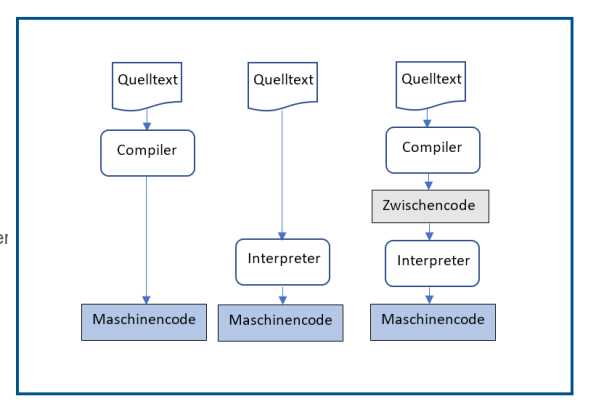
Vom Quelltext zum ausführbaren Programm



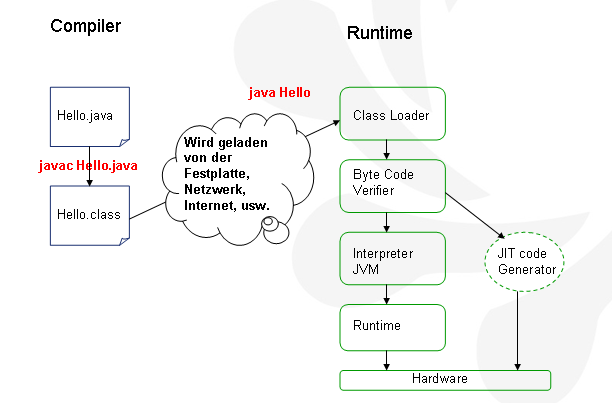
*JavaScript*

*Assembler, C, C++*

*Java, C#, Python*

Java Basics

1. **Compiler, Interpreter, Classpath, Packages**

****

The compiler creates the so called Byte Code, executable on the Java Virtual Machine (JVM).  
The “Byte Code” is a kind of assembler code for the JVM. For each supported platform (Windows, Solaris, Mac, …) exists a dedicated JVM which can interprete and execute this Byte Code. That means: ones the java source code has been compiled, the resulting byte code will run on any supported operating system.

JIT (Just In Time) Code Generator:  
For speed optimization, frequently executed byte code (hot spot) will be recognized by the JVM and transferred directly into native code (platform dependent).  
This native code remains in memeory during the actual JVM instance is alive or with other words: as long as your current program runs.

* 1. **A Simple Java Program**

class Hello {

public static void main (String [] args) {

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

Greeting g = new Greeting();

g.sayHello();

}

}

class Greeting {

void sayHello() {

System.out.println("hello, how are you?");

}

}

Output:

Hello World!

hello, how are you?

* 1. **Compile and Run Java Programs from the Command Line**

General form Compiler:

(box brackets “[..]” are optional)

**javac [options1 option2] [path]ClassName.java**

The following will compile all files in this folder (even if not changed)

**javac \*.java**

General form Interpreter JVM:

(box brackets “[..]” are optional)

**java [options1 option2] ClassName [argument1 argument2 …]**

**Example 1:**

The *Hello.java* and the *Greeting.java* Classes are located in one folder:

*HOME\GettingStarted\compilerTestSimple*

**Open the Console from** HOME\GettingStarted\compilerTestSimple  
  
Starting the Compiler: javac Hello.java  
Product: Hello.class, Greeting.class  
Running Hello.class: java Hello

**Example 2:**

The *HelloX.java* and the *Greeting.java* classes are located in two different folders:

*HOME\GettingStarted\classPathTest\CompilerTest* (HelloX.java)

and

*HOME\GettingStarted\classPathTest\CompilerTest2* (Greeting.java)

**Open the Console from** HOME\GettingStarted\classPathTest   
*the following will not work! why?*

javac CompilerTest\HelloX.java

the problem is that the class Greeting.java cannot be found

*To compile the program, you need to set the class path for* ***every*** *classes used by the main class!*   
javac –cp CompilerTest2 CompilerTest\HelloX.java

Option Flag Class Path: -cp   
or : -classpath

Product: HelloX.class, Greeting.class

*To run the program, you need to set the class path for* ***all*** *classes (also for the main   
 class)! Separated by semi colons ( ; )*  
  
Running HelloX.class: java -cp CompilerTest2;CompilerTest HelloX

**Open the Console from** HOME\GettingStarted\classPathTest\CompilerTest   
(the folder, where HelloX.java is located)

javac -cp ..\CompilerTest2 HelloX.java

java -cp .;..\CompilerTest2 HelloX

the DOT means the actual folder including the HelloX main class

The class path is a list of folders (separated by semicolons), containing classes which might be used by the currently executed programm. In that case the class loader will load them on demand.

**Example 3 –Creating and using JAR archives and executable JARs**

Open the Console from HOME\GettingStarted\jar

From the console we use the jar tool provided by JDK

**Creating executable jar files**

Name of resulting jar

*jar cmfv* ***main.txt*** *MyPrj.jar \*.class*

Resources to be zipped. Here all class files in this folder

Manifest file **main.txt** contains the main class name

Main-Class:Hello

**run the executable jar MyPrj.jar**

*java -jar MyPrj.jar*

**Creating just an archive including only the Greeting class**

*jar cfv MyGreeting.jar Greeting.class*

**compile Hello.java using the archive MyGreeting.jar**

*javac -cp MyGreeting.jar Hello.java*

**execute Hello.class using the archive MyGreeting.jar**

*java -cp .;MyGreeting.jar Hello*

**Remark**

In case of archives, the class path must include not only the folder but also the archive name  
inclusive the extension *.jar*

**Other useful commans:**

**unpacking the jar content** *jar xf MyPrj.jar*

**listing up the jar content** *jar tf MyPrj.jar*

**Example 5 – Using the Destination flag: -d** (Compiler only)

With the destination flag **–d** we can define a folder where the compiler should store resulted   
 class files instead mixing them all together with the source files

**Open the Console from** HOME\GettingStarted\compilerTestSimple

javac **-d classes** Hello.java

java -cp classes Hello

**Example 6 – Using the Property flag: -D** (Execution only)

With the property flag we can pass one or more system property values to the application, when we start the JRE.   
Syntax: java *–Dname1=value1 –Dname2=value2* MainClassName

Example:

public class PropertyTest {

public static void main (String [] args) {

System.out.println(

"The color is: "+ System.getProperty("color"));

}

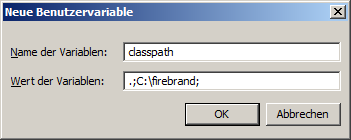
}

**Open the Console from** HOME\GettingStarted\propertyTest

Start the JRE: java -Dcolor=red PropertyTest

* 1. **Setting Class Path as System Environment Variable in Windows:**

Start->Einstellungen->Systemsteuerung->System->Erweitert->Umgebungsvariablen   
-> Neu



Always add the actual path to the class path using a **dot (.)**   
This means:   
to which ever folder the console points to, its contents are added to the class path.

* 1. **Packages and Imports**

Package names represent folder names. Source files with package names in their source code   
 must be located in the named folder.  
 If a class uses a class from a different package, it must import it.

Only classes having a package name can be imported!

Classes of package java.lang are always imported by the compiler.

package p2;

public class Greeting {

public void sayHello() {

System.out.println("hello, how are you?");

}

}

--------------------------

package p1;

import p2;

class HelloX {

public static void main (String [] args) {

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

Greeting g = new Greeting();

g.sayHello();

}

}

**Open the Console from** HOME\GettingStarted\packages

This will create class files directly into folders: p1 and p2

javac p1\HelloX.java

Run the main class:

java p1.HelloX

the JRE needs the fully qualified name of the main class which is package name and class name separated by a dot (.)   
**The console must point to the parent folder of the main class folder.**  
**Setting the class path to the parent folder of the main class folder does the trick, when starting** p1.HelloX **from another position:**

**Open the Console from** HOME\GettingStarted   
java -cp packages p1.HelloX

**package names and their usage:**

For example in: test\myclasses\Greeting.java  
Greeting.java could have the following package definition:

package test.myclasses;

*The folder Structure “test\myclasses” is represented by the package name “test.myclasses”*

The import statement in Hello.java would have to be:  
import test.myclasses.Greeting;

or

import test.myclasses.\*; indicating that all classes in this package are imported

*Remark: Asterix (\*) means: import all classes from the package. It cannot be used for sub-package names. Only for classes within the indicated package!*

But we can also use the fully qualified class names alone, without imports:

For example:

class Hello {

public static void main (String [] args) {

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

*test.myclasses.Greeting g = new test.myclasses.Greeting();*

g.sayHello();

}

}

Sometimes this can be helpful, if two or more classes have the same name in different packages. In order to tell the compiler which implementation of the class should be used, we need the fully qualified class name at the point where the class is involved.

**Static Imports**

This feature simplyfies the access to static members of classes   
i.e.: Math.PI or Math.sqrt(double d)  
Instead of always writing in style: “className.staticVarName” we simply import the static members of this class and can access directly its members.

Example:

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

import **static** java.lang.System.**out**;

class StaticImport {

public static void main(String[] args) {

**out**.println(**PI**);

}

}

*Remark: Asterix (\*) means: import all static members (Variables and methods) of the indicated class.*

In order to import static methods like Math.sqrt(double d) we simply have to write the method name only: **import static java.lang.Math.sqrt;**