How to create, compile and execute a Java program

The Java development process consists of several steps.

1. Create the Java sourcecode and save it in a file,
2. Compile the sourcecode to bytecode,
3. Execute and test the bytecode.

Experience shows that even simple programs quite often contain syntactic or logical errors which have to be corrected. So the complete development process usually has to be repeated several times to eliminate all compiler and execution warnings or errors.

1. Create Java sourcecode

The smallest unit of a valid Java sourcecode is an entity named **`class`** (the exact meaning of this keyword will be explained in a later chapter). Java sourcecode normally contains only one **`class`**, but for training purposes several classes can be stored in one single file – just for convenience. In each case the file name cannot be chosen arbitrary, but must

- match exactly the name of the `class` which includes a valid **`main( )`** method
- have the extension "`.java`", e.g. **`Test2.java`** (see Example 2).

For writing and editing any convenient editor may be used – for Linux systems e.g. Geany, for Windows UltraEdit. A platform-independent IDE (integrated development environment) like Eclipse or NetBeans is well suited for bigger Java projects mainly because it offers syntax highlighting and additional sophisticated correction tools. In any case it is preferable if the editor offers automatic Java compiling and executing features so that the editor has not to be terminated to run the Java compiler.

```
// Example 2: Test2.java

class Test2 {

        public static void main (String [] args) {

             System.out.println ("Just a Test ...");
        }
}
```

2. Compile Java sourcecode to bytecode

Java sourcecode is compiled either directly by calling "**`javac Test2.java`**" from the command-line or by letting the IDE or editor do this. In the course of the compilation process an automatic spell- and syntax-check is done. If errors or warnings occur they will be displayed and the compilation process terminates. If everything is correct the Javac compiler displays no output.

```
javac Test2.java
```

After successful compiling, a class file is created which contains the bytecode - ***`Test2.class`***.

Additional information: If the sourcecode contains several classes then each class will be compiled to a separate file. So after compilation there may exist a number of different class files. If the class files are already compiled, the compiler will check if they are up to date (compared to the corresponding source code). If this is not the case, they are recompiled .

Since each class in the sourcecode is transformed by the compiler into exactly one class file, it is ultimately irrelevant whether the Java source code was present in a single file or in multiple files. The decision is based on practical considerations (length of the source code, number of classes ...).

3. Execute Java bytecode

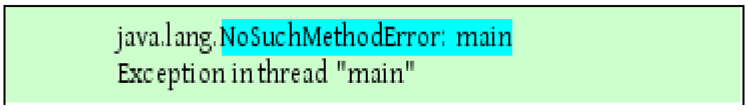
As already explained, Java bytecode does not contain any directly executable processor instructions and therefore it cannot be started by the operating system. To run a Java application, a platform-specific Java interpreter (also called Java Virtual Machine JVM, or Java Runtime Environment JRE) must be present on the local computer. This VM in fact itself is a local application which analyses the bytecode line by line and converts it (in runtime) to executable real machine instructions. This real machinecode is not stored anywhere, but is executed in the internal memory of the Java virtual machine – and it is lost when the Java VM terminates.

The Java interpreter is invoked from the command line by calling **`java`** added by the name of the class file. The extension **`.class`** must be omitted:

```
java Test2
```

What happens is that the Java VM looks into the class file specified and searches for a start method named **`main()`**. If this method is present, its statements are executed. When more classes are needed, then the interpreter searches for the additional class files. This always assumes that all the Java classes associated with the project were compiled (step 2), because the Java interpreter can not compile, but only execute byte code.

If the main () method is NOT found, the error message appears:



```
java.lang.NoSuchMethodError: main
Exception in thread "main"
```

This error message indicates that the specified class file does not contain a method named **main ().** In this case, the name of the class file and the sourcecode has to be checked for typographic and syntactic errors.