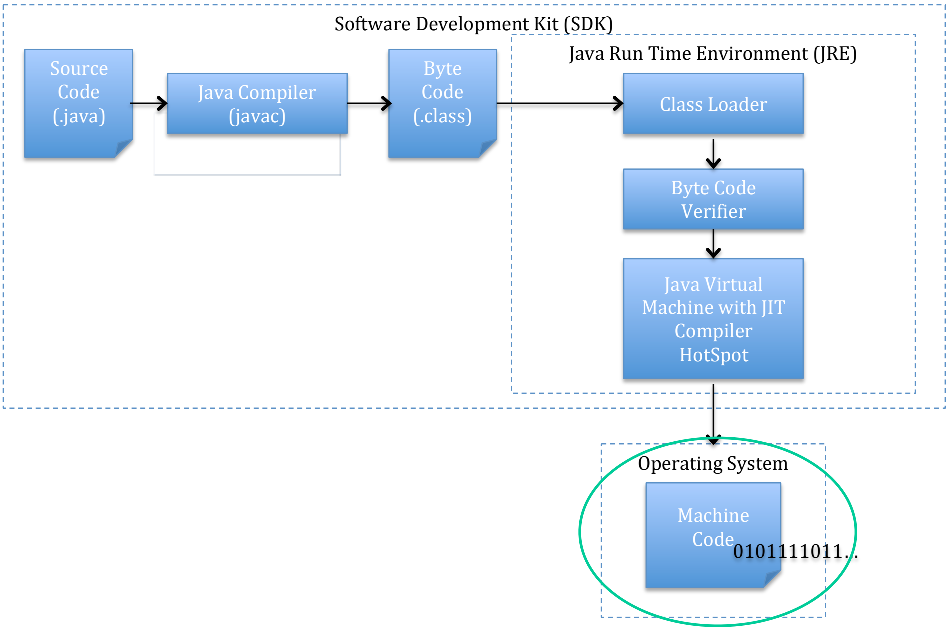
The Java architecture

Advanced Programming

The Java architecture - overview



Software Development Kit (SDK)

Source Code (.java)

Java Compiler (javac)

Byte Code (.class)

Java Run Time Environment (JRE)

Class Loader

Byte Code Verifier

Java Virtual Machine with JIT Compiler HotSpot

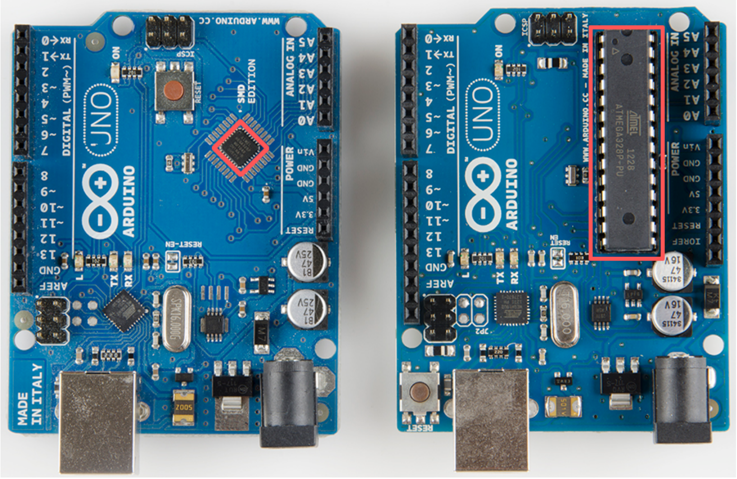
Operating System

Machine Code

0101111011...

Central Process Unit

- Central Processing Unit (CPU) (or simply processor) executes bits operations



The machine code

- A machine code is a set of bits operations executed by the CPU
- Instructions are **patterns of bits** that by physical design correspond to different commands to the machine

Example of machine instruction

(R-type operation)
Adding register 1 and 2 and placing to register 6; “funct” describes the operation

```
[  op  |  rs |  rt |  rd |shamt|funct]
    0     1     2     6     0     32     decimal
 000000 00001 00010 00110 00000 100000  binary
```

Machine code is not that readable for human being

- ☆ You can learn more in the Architecture of Digital System course!

Machine code is not portable

- Systems may differ in some details, e.g.,
  - memory arrangement,
  - operating systems, or
  - peripheral devices
- Thus, systems will not run the same machine code even when the same type of processor is used

Machine code is not portable

- Every processor or processor family has its own machine code instruction set
- Each processor can manage one or more cores

Compilation

- Machine code is not portable and human beings may have hard time to read it
- Higher level languages allow to write code that humans can better understand, but they have to be translated into machine code
- Translation has two forms: compilation and interpretation

Compilation

- Compiler: faster
  - Read the code once and prepare it to be executed by the given CPU
  - Good compilers can perform optimisations
- Interpreter: portable
  - It reads each code line by line and translates end executes it every time
  - It is independent from the CPU implementation (e.g., it does not need to store any code prepared for the CPU)

Example - Interpretation

- Example: how can the following statement be executed?
  - A[i][j] = 1;
- Create a software environment that understands the language (in this case the 2-dimensional array)
- Run the statement: just put 1 in the array entries

Interpretation

- This is **interpretation** since the software environment understands the language and performs the operations specified by interpreting the statement in one shot
- If the statement appears another time the interpreter repeat the whole process

---

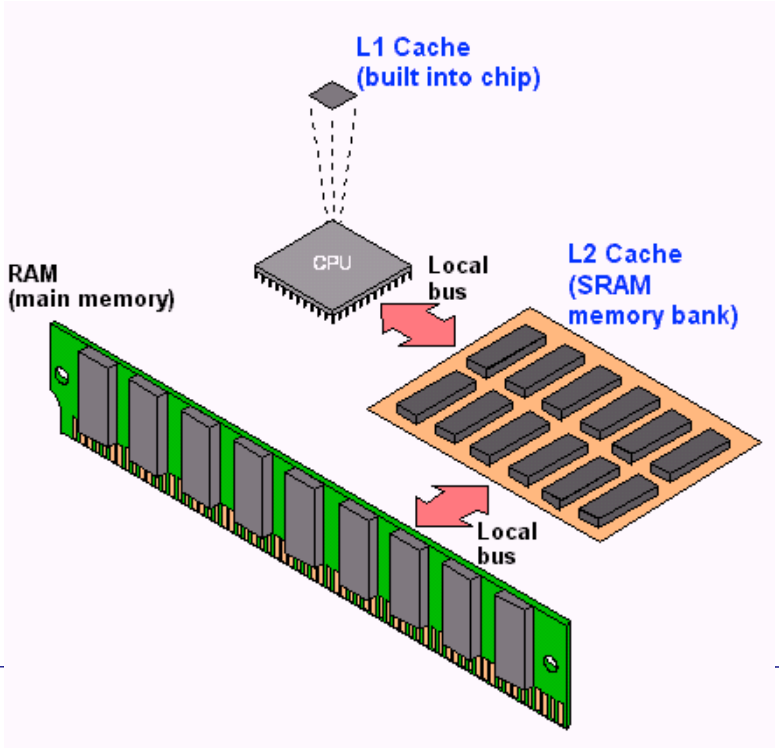
Example - Compilation

- Example: how can the following statement be executed?
  - A[i][j] = 1;
- Translate the statements into native machine (or assembly language) and then run the program on the CPU

Compilation

- A good compiler performs a set of optimisations:
  - for instance, while it looks up for a variable in the RAM (main memory), it performs other CPU operations on the rest of the machine code until the variable has been retrieved
- This is possible because the whole code as been translated into machine code and the compiler needs not to translate it again

Storage hierarchy in computer



Example

- An **if** statement is always an **if** statement each time it's encountered, so that analysis can be done once.
- Which branch will be taken depends on the runtime value of an expression
- The compiler emits code to test the value of the expression and take the appropriate branch
- The execution will decide which branch on the runtime value of an expression

---

Compilation and interpretation

- In general, which approach is more efficient?
  - A[i][j] = 1;

Compilation:

```
    salq    $2, %rax
    addq    %rcx, %rax
    leaq    0(,%rax,4),
    %rdx
    addq    %rdx, %rax
    salq    $2, %rax
    addq    %rsi, %rax
    movl    $1, A(,%rax,
    4)
```

Interpretation:

create a software environment
that understand the language
put 1 in the array entry A[i][j];

Interpretation

- Interpreting a code cannot perform much optimisation as it knows only the instructions for part of the code until the line that has been interpreted
- To interpret a code in a specific language, we need to have an environment that compiles and executes (both!) the code line by line
- Thanks to its environment the language is **portable**

Interpretation - portability

- An example are PHP or Python and many scripting languages (e.g., Java script)
- Example: when we call "php -file.php” from a shell command, we are invoking such environment
- The good aspects of such environment is that they can be easily used in web browser

Interpretation

- Programming languages can be compiled or interpreted or both
- The Java architecture is a good example to see how compilation and interpretation work together

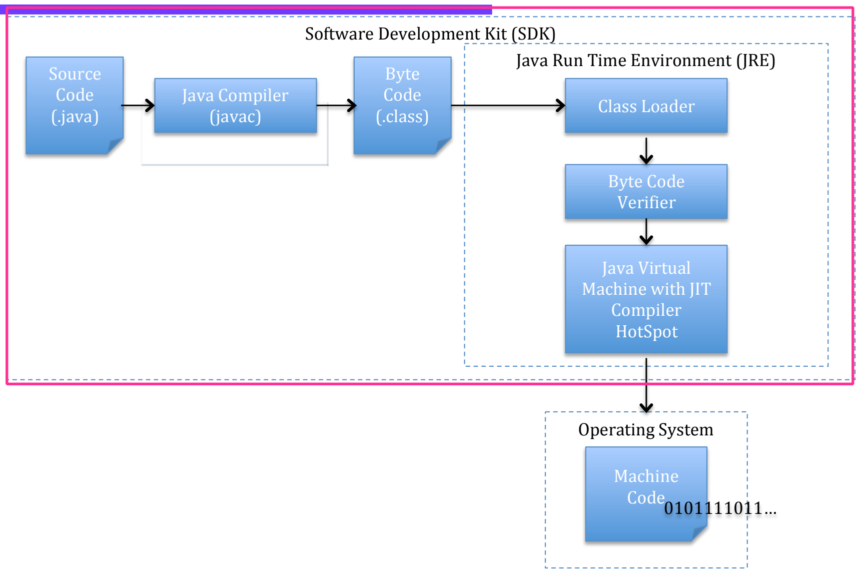
Java Standard Edition platform- JSE

- JSE is a set of specifications for a software environment in which a Java program is compiled and runs
- It consists of several components

Java Standard Edition platform- JSE

- Every implementation of JSE platform must adhere to the following specifications
  - Development tools to compile, run, monitor, debug, and document an application
  - Application Programming Interface - API
  - Deployments tools
  - User Interface Toolkits
  - Integration Libraries

The Java architecture



Software Development Kit (SDK)

Source Code (.java)

Java Compiler (javac)

Byte Code (.class)

Java Run Time Environment (JRE)

Class Loader

Byte Code Verifier

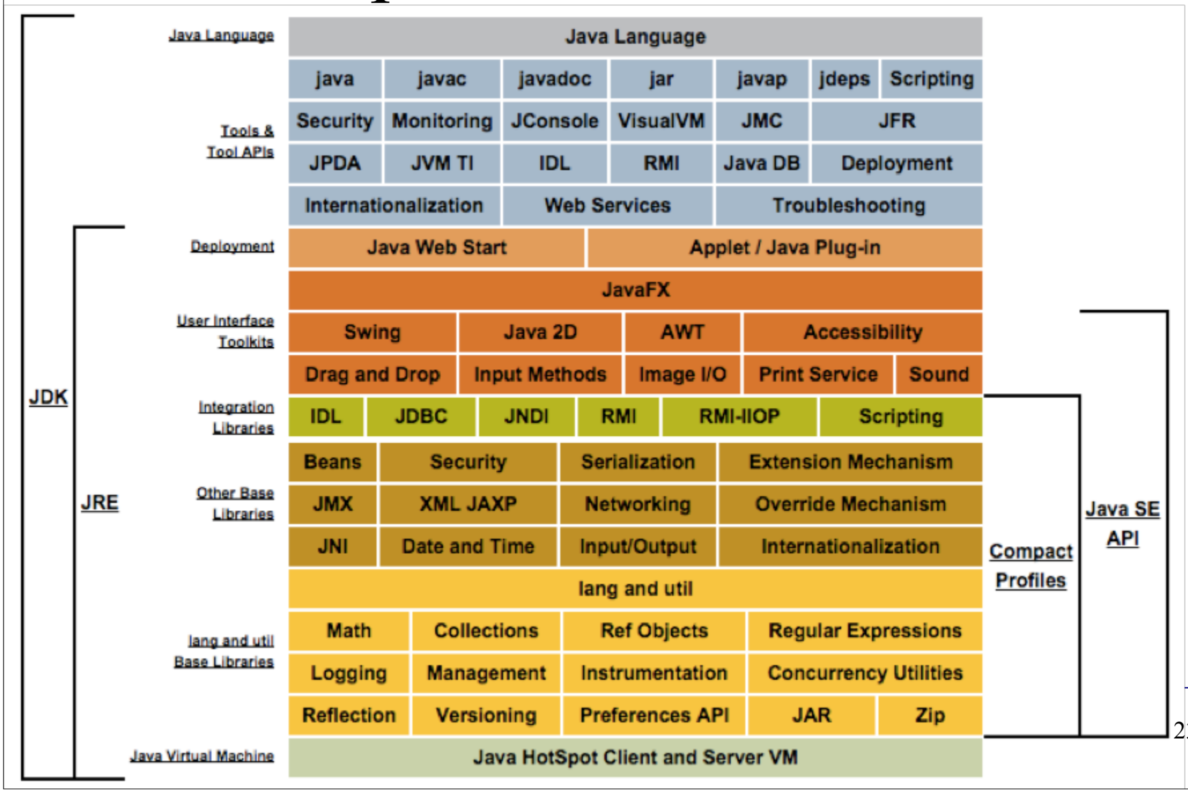
Java Virtual Machine with JIT Compiler HotSpot

Operating System

Machine Code

0101111011…

Oracle implementation of JSE



Java Language	Java Language
Tools & Tool APIs	java, javac, javadoc, jar, javap, jdeps, Scripting Security, Monitoring, JConsole, VisualVM, JMC, JFR JPDA, JVM TI, IDL, RMI, Java DB, Deployment Internationalization, Web Services, Troubleshooting
Deployment	Java Web Start, Applet / Java Plug-in
User Interface Toolkits	JavaFX Swing, Java 2D, AWT, Accessibility Drag and Drop, Input Methods, Image I/O, Print Service, Sound
Integration Libraries	IDL, JDBC, JNDI, RMI, RMI-IIOP, Scripting
Other Base Libraries	Beans, Security, Serialization, Extension Mechanism JMX, XML JAXP, Networking, Override Mechanism JNI, Date and Time, Input/Output, Internationalization
lang and util Base Libraries	lang and util Math, Collections, Ref Objects, Regular Expressions Logging, Management, Instrumentation, Concurrency Utilities Reflection, Versioning, Preferences API, JAR, Zip
Java Virtual Machine	Java HotSpot Client and Server VM

Java tools - shell commands

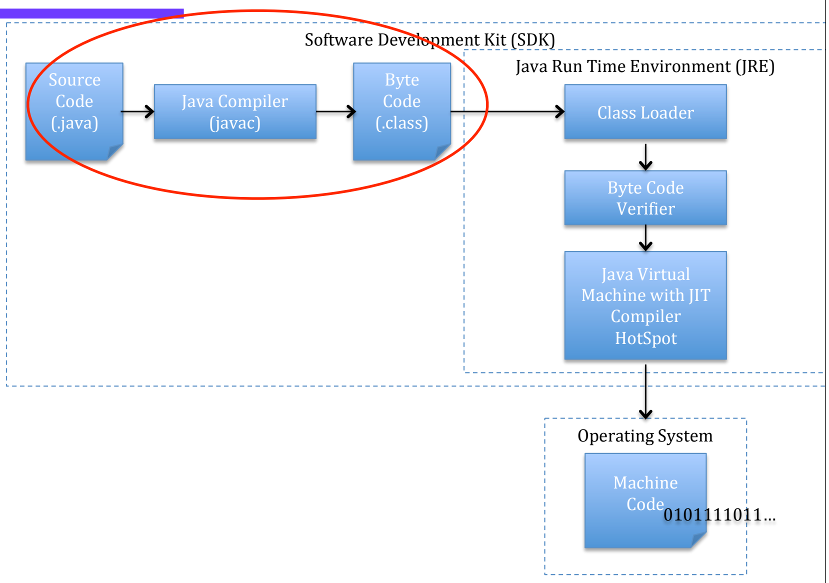
- **javac** - Java compiler; **javac foo.java**
- **java** - Java Application launcher; **java foo 1 12 3 4**
- **jdb** - Java debugger; **jdb foo 1 12 3 4**
- **javadoc** - Java API doc. generator; **javadoc**
- **jar** - java archive tool; **java -jar foo.jar**
- **javap** - Java class file disassembler; **javap foo.class**

Example with javadoc

- My project contains the package **com.test** and I want to put the generated documentation in files located in a specific folder like this: **C:/javadoc/test**

- javadoc -d C:/javadoc/test com.test
- javadoc -d [path to javadoc destination directory] [package name]

Java compilation



Software Development Kit (SDK)

Source Code (.java)

Java Compiler (javac)

Byte Code (.class)

Java Run Time Environment (JRE)

Class Loader

Byte Code Verifier

Java Virtual Machine with JIT Compiler HotSpot

Operating System

Machine Code

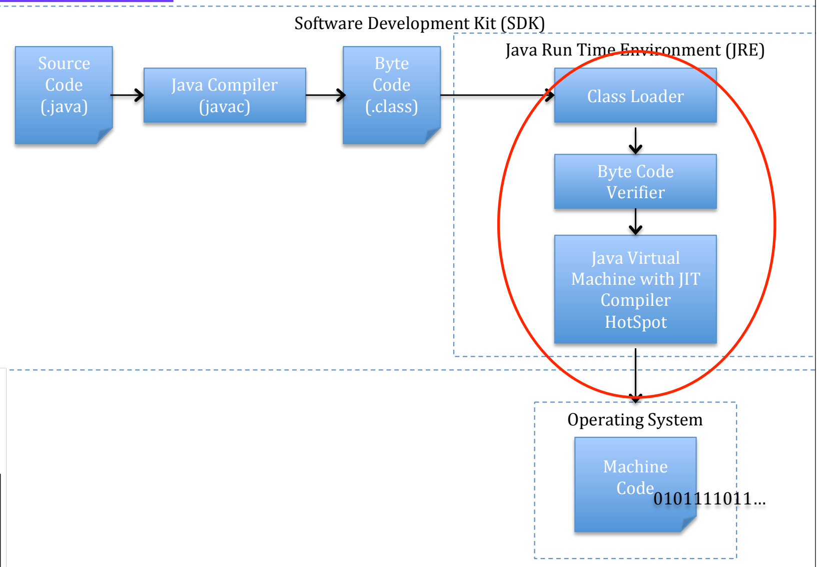
0101111011…

Java compilation

- In Java, we can simply write the source code in plain text files ending with the .java extension
- With the help of the java **compiler (javac)**, source files are then compiled into class files ending with the .class extension

---

The Java Run-time Environment


Software Development Kit (SDK)

Source Code (.java)

Java Compiler (javac)

Byte Code (.class)

Java Run Time Environment (JRE)

Class Loader

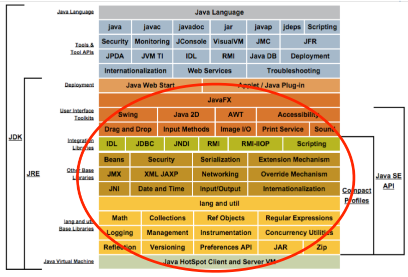
Byte Code Verifier

Java Virtual Machine with JIT Compiler HotSpot

Operating System

Machine Code

0101111011…



JRE

- The implementation of a JVM performs the following actions
  - Load,
  - Link,
  - Resolve, and
  - Initialise classes and interfaces

Executing

- A **.class** file contains bytecode written in the language understandable by the Java Virtual Machine (JVM)
- The byte code has a hardware-and-operating-system-independent binary format
- The class **loader** dynamically reads binary names of .class files (i.e., package name and class name like java.lang.String) at run-time and search for these names in the file system

Loading- class loader

- Find the binary representation of a class or interface type with a particular name and create a class or interface file from that binary representation
- If such class file is not found the the loading throws the exception: **ClassNotFoundException**

Linking

- Linking verifies and prepares the correctness of a type so that it can be executed.
  - **Preparing** allocates memory for the static field of classes and interfaces and initialise them to their default values
  - **Verifying** ensures that the binary representation of a class is structurally correct

Resolving

- Resolving associates references to run-time values

Initialising

- Initialise classes and interfaces
- Initialisation defines the values of class or interface variables

Java Virtual Machine - JVM

- The JVM is an abstract entity that is instantiated for different operating environments (Microsoft Windows, Solaris Operating System, Linux, or Mac OS)
- The JVM is a "virtual machine" : it aims at emulating the machine (CPU) **interpreting the byte code**
- The SUN JVM http://hg.openjdk.java.net/jdk8

JVM

- As the JVM is available on many operating systems, the same .class can be translated into machine code on different OSs
- Interpreters have been considered slow, but Java interpreter runs compiled byte-code and as such it is a relatively fast
- Last versions of JVM have adopted JIT compilation that make JVM even faster

Just-In-Time (JIT) compilation

- The JIT translation consists of having a compiler which generates code for an application (or class library) during the execution of the application itself
- JIT cannot be performed on the whole byte code

Adaptive compilation

- Programs spend almost all of their time executing a **relatively small part** of the code again and again
- The chunk of code that is executed repeatedly may only be a small percent of the total program, but its behaviour **determines the program's overall performance**

Adaptive compilation

- The adaptive compilation first **profiles** the code while it is executing, to see what parts are being executed repeatedly
- Once it knows it, it translates only these sections into machine code

---

Adaptive compilation

- Since it only compiles a small portion of the byte code, it can afford to take the time necessary to optimise those portions
- The rest of the program may not need to be compiled at all - just interpreted - saving memory and time (JIT)

The JIT HotSpot VM

- HotSpot http://openjdk.java.net/groups/hotspot/ is a JVM that uses the JIT translation and adaptive compilation
- HotSpot implements the JVM Specification, it is written in C/C++ and is delivered as a shared library in JRE

The JIT HotSpot VM

- It allows the interpreter time to "warm up" Java methods, by executing them thousands of times
  - The first execution of the program can be a bit slower
- It uses a configurable invocation-count threshold to decide which methods to compile

Compiling and executing

- The java **loader** runs the file with an instance of the JVM, specific for each CPU and operating system
- Before passing the class file loaded in the run-time environment, the Java **verifier** checks the fields for security