Java CLI JDK command line

CONCEPTS

> Source becomes bytecode with javac, bytecode runs on the JVM with java. Tools like jar, jshell and jcmd all ship in the JDK.

Main.java source
↓ javac
Main.class bytecode
↓ jar
app.jar archive
↓ java -jar
JVM running app

> Class path separator is : on Linux and macOS, ; on Windows.

RUN

`java -version`
Print version and exit

`java <File>.java [args]`
Compile in memory and run

`java -cp <path> <MainClass> [args]`
Run a class

`java -jar <file.jar> [args]`
Run a JAR's main class

`java -Dkey=value ...`
Set a system property

`java -ea ...`
Enable assertions

`java -m <module>/<MainClass>`
Run a module

CLASS PATH & MODULES

`-cp "lib/*:out"`
Add all JARs in lib, plus out

`--module-path <dir> / -p <dir>`
Where application modules live

`--add-opens <mod>/<pkg>=ALL-UNNAMED`
Open a package to reflection

MEMORY & JVM FLAGS

`-Xmx<size>`
Maximum heap

COMPILE (JAVAC)

`javac <File>.java`
Compile one file

`javac -d <dir> <files>`
Put class files in dir

`javac -cp <path> <files>`
Compile against libraries

`javac --release <n> <files>`
Target a Java version

PACKAGE (JAR)

`jar --create --file <f.jar> -C <dir> .`
Pack a directory

`jar --create --file <f.jar> --main-class <Class> -C <dir> .`
Runnable JAR

`jar -tf <f.jar>`
List contents

JSHELL

`jshell`
Start the REPL

DIAGNOSE

`jcmd`
List running JVMs

`jps -l`
PIDs with main class or JAR

`jcmd <pid> Thread.print`
Thread dump

`jcmd <pid> GC.heap_dump <file>`
Write a heap dump

ENVIRONMENT

`JAVA_TOOL_OPTIONS`
Options for every JVM