COMP6700/2140 JDK Tools

Alexei B Khorev and Joshua Milthorpe

Research School of Computer Science, ANU

February 2017

Topics

1. JDK tools (most important)
2. Their use

GDK Tools

We've learned about tools of programming — editors and IDEs. How to proceed whence the text of your program is complete, and one needs to compile and execute it?

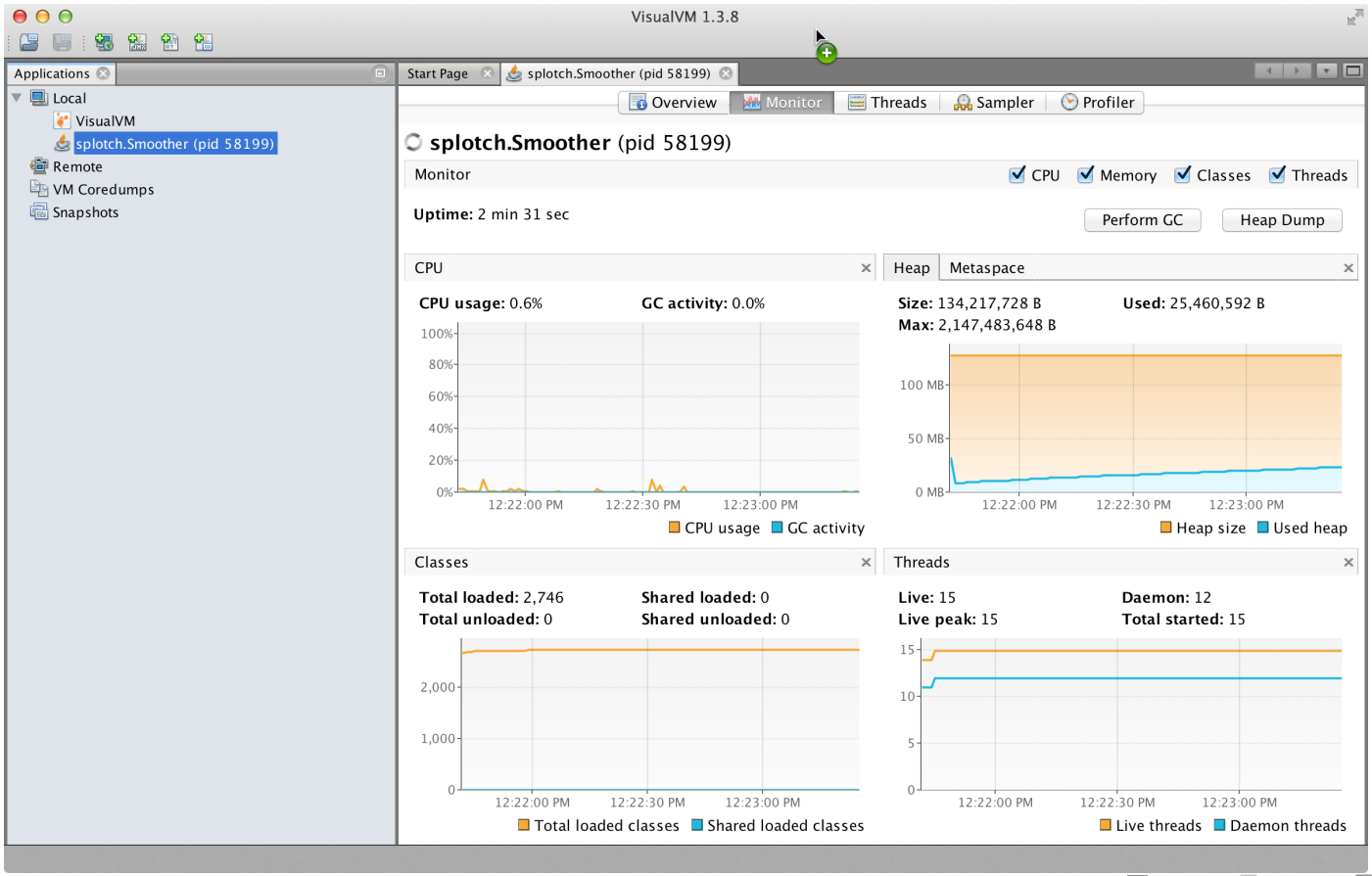
Java [Software] Development Kit, *JDK*

The Java programming environment includes a set of programs (*tools*), which is needed by everyone who wants to create software in Java. The list of these tools is ever growing (Oracle's Java SE 8 has 44). The most important ones are:

- `javac` — Java *compiler*, usage : `javac MyProgram.java`; output : `MyProgram.class` (the byte code). Different Java compilers exist (eg, *incremental* Eclipse's Compiler for Java, `ecj`)
- `java` — Java *interpreter*, usage : `java MyProgram`; output : running the program
- `javadoc` — Java *documentation generator*, usage: `javadoc MyProgram.java` ; output : a bunch of `html` files
- `jdb` — Java *debugger*
- `jar` — Java *archiving tool* (for distributing)
- `appletviewer` — Java *applet viewer* (has little use nowadays)
- `javap` — Java *class file disassembler*
- various diagnostic tools: `jps`, `jstat`, `jstack`, `jconsole`, `jhat`

How JDK tools are used

They all can be used on the command-line. Some editors (*eg*, TextMate) can use them directly (`javac` and `java`). IDEs can use almost all. Some JDK tools are built-in in specialised application like *VisualVM* or *Java Mission Control*, which are used for Profiling, Monitoring, and Diagnostics (how much memory an application uses, how many classes are loaded, how garbage collector operates and so on). IDEs (with special plugins) can also do such work.



Use of javac

The program (with a compile error)

```
public class Arithmetic {
   public static void main(String[] args) {
      assert args.length > 1;
      int x = args[0];
      int y = args[1];
      System.out.printf("%10s = %3d%n", "x + y", x+y);
      System.out.printf("%10s = %3d%n", "x - y", x-y);
      System.out.printf("%10s = %3d%n", "x * y", x*y);
      System.out.printf("%10s = %3f%n", "(x + y)/2", x/y);
   }
}
```

when compiled:

```
% javac Arithmetic.java
```

```
Arithmetic.java:4: error: incompatible types: String cannot be converted to int   int x = args[0];
                                                  ^
Arithmetic.java:5: error: incompatible types: String cannot be converted to int   int y = args[1];
                                                  ^
2 errors
```

Use of java

Once corrected and compilation succeeds — the bytecode `Arithmetic.class` is generated. It can be executed, but the result is a run-time error:

```
% java Arithmetic 12 7
     x + y =  19
     x - y =   5
     x * y =  84
 (x + y)/2 = Exception in thread "main" java.util.IllegalFormatConversionException:
 ... ... // full exception stack is printed here
 at Arithmetic.main(Arithmetic.java:9)
```

More fixes are required to eliminate run-time errors.

`javac` and `java` do not just compile and run your program, they help you to find and fix errors (**bugs**).

Use of javap

How to see inside a class, if only its bytecode is available?

If a bytecode has been compiled from the source:

```
public class MembersInAClass {
    public String name;
    public int nameLength() { return name.length(); }
    public MembersInAClass(String name) { this.name = name; }
}
```

One can see all members of *MembersInAClass* by running javap on the bytecode:

```
% javap MembersInAClass
Compiled from "MembersInAClass.java"
public class MembersInAClass {
  public java.lang.String name;
  public int nameLength();
  public MembersInAClass(java.lang.String);
}
```

running it with the option `javap -c` will show the bytecode (instructions for JVM) itself. Who needs the source? (☺)

Use of javadoc

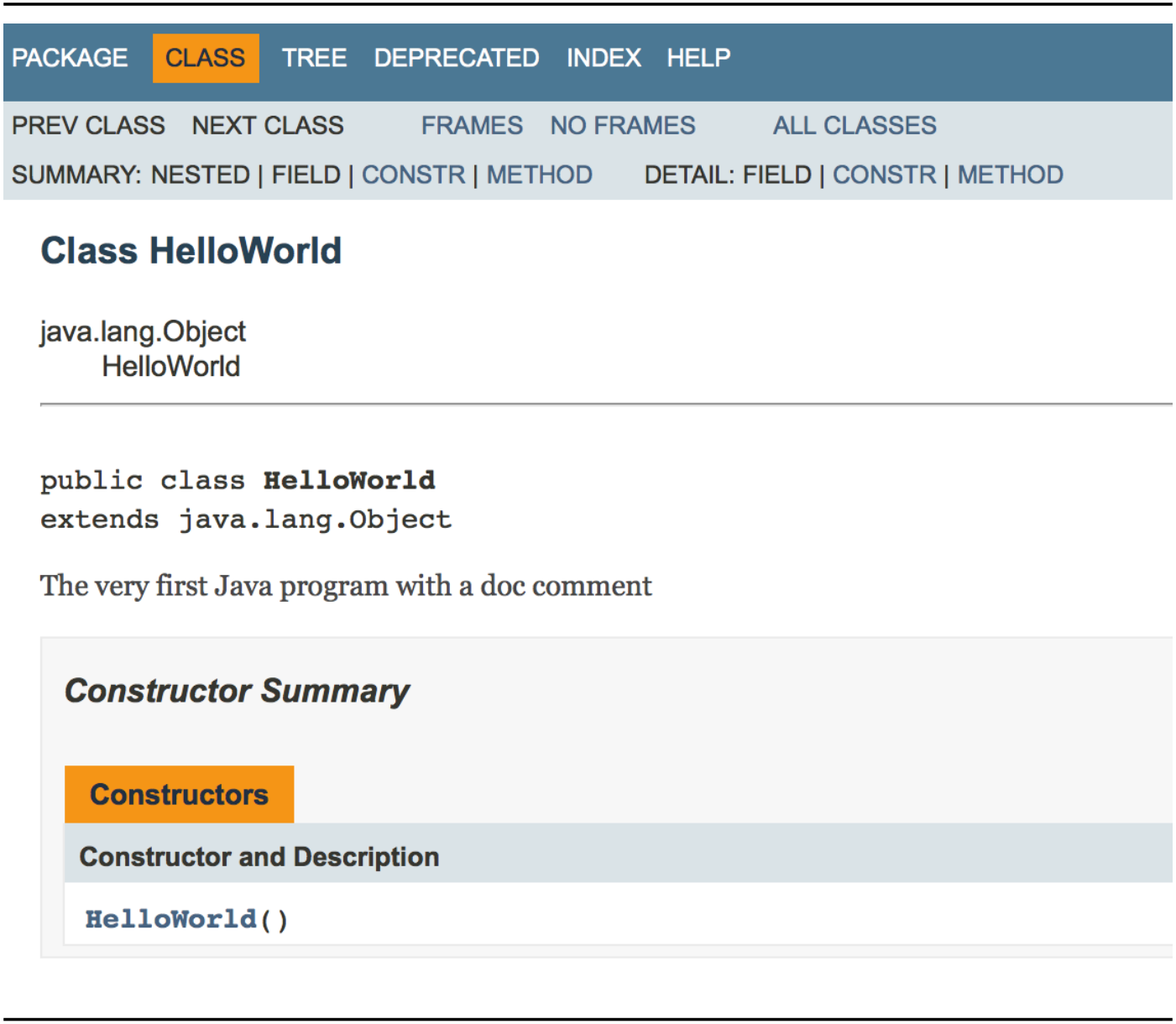
A code with *doc-comments* can produce (nice) documentation with javadoc:

```
/** The very first Java program with a doc comment
 *  @author abx
 *  @version 1.0
 */
public class HelloWorld {
    public static void main(String[] args) {
        // the inline comment -- will be ignored by javadoc
        System.out.println("Hello World");
    }
}
```

```
% javadoc HelloWorld.java
Loading source file HelloWorld.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_72
Building tree for all the packages and classes...
Generating ./HelloWorld.html...
...
```

Open `HelloWorld.html` in a browser and enjoy the show...

Generated Documentation



PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class HelloWorld

java.lang.Object
    HelloWorld

```
public class HelloWorld
extends java.lang.Object
```

The very first Java program with a doc comment

Constructor Summary

Constructors

Constructor and Description
`HelloWorld()`

Where to look for this topic in the textbook?

The use of these and other tools can be further studied in

- JDK Tools and Utilities