COMP6700/2140 Tools of Programming

Alexei B Khorev and Joshua Milthorpe

Research School of Computer Science, ANU

February 2017

Topics

1. Tools to create code
2. Tools to build an executable
3. How to execute
4. **I**ntegrated **D**evelopment **E**nvironments – IDEs
5. Tools in action

Tools of Java programming — 1

A program is just a text with instructions. How to make computer to execute them? And how best to write programs? Use *tools* — special applications to create software.

- *Baby tools* : a text editor (better with the syntax support & colouring and automatic layout) + SDK commands run on the terminal (console). For the SDK commands see later. The available editors are:
  - Kate or Gedit (Linix), TextMate (Mac), SublimeText or Atom (crossplatform), emacs or vim (also cross-platform, but for hackers), and many more...
- Mini *IDE* — **I***ntegrated* **D***evelopment* **E***nvironment*, a lightweight version of a *real* IDE with many functions and capabilities not present. Good for learning, not for professional work. Two useful mini-IDEs: *DrJava* and *BlueJ*.
- Proper IDE: These are formidable tools. All are quite similar in what they can do. The difference is in *how* they do it, and how their capabilities can be extended. The difference is important depending on the kind of work which you are doing, but this has to be a *professional* work. Require a steep learning curve. Benefits are enormous. Even a novice can get great benefits. The IDEs which are currently popular are:
  - Eclipse, NetBeans, IntelliJ IDEA (all available in labs)

  Eclipse and Netbeans are free (both available in labs), IntelliJ has a slim-down *Community Edition* free version. I (and many others) consider IDEA the best Java IDE.

Tools of Java programming — 2

Java Software Development Kit, *SDK*

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*
- `appletviewer` — Java *applet viewer*
- `javap` — Java *class file disassembler*
- various diagnostic tools: `jps`, `jstat`, `jstack`, `jconsole`…

Programming Plain Text Editor

Choose:

- TextMate (on Mac only)
- Sublime Text
- Gedit (installed in student labs)
- Atom (installed in student labs)
- LightTable
- NightCode

or whatever you like: it just needs to be a **plain text programming editor** with:

1. syntax highlighting
2. automatic formatting (indentation)
3. name completion
4. code insertion triggers (activated by "hook-and-tab")
5. macros recording
6. efficient find/replace capability
7. et cetera, et cetera

if your editor doesn't have some of those – never mind, 1 and 2 are already good enough.

TextMate

HelloVoter.java — lecturettes

```
/**
 * Execution of Dave Lister for stealing apple of the people
 */

import java.util.Scanner;

class HelloVoter {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("What is your name, voter? ");
        String name = sc.nextLine();
        System.out.
            printf("Kneel, voter %s, you are under the sentence of death!%n", name);
    }

}
```

Line: 14:76 | Java | Tab Size: 4 | main(String[] args)

Command-line

Power through abstraction

The operating system command-line interface (accessible via `Terminal` application or the like) is not a backward primitive computer interface (like Windows user might think). It has been and still is the most powerful and productive way to communicate with computer and make it to do one's bidding.

In the beginning was the Command Line

(Neal Stephenson)

Main SDK commands

- `javac` – to compile readable code into bytecode (converts `.java` file to `.class` file):

  % javac HelloVoter.java ↩

  Terminate every command with the return key ↩ pressing; % is a command-line prompt, it may look different on your computer. If no compile errors occur, this will generate the bytecode file (class) `HelloVoter.class`
- `java` – to launch JVM and execute the bytecode of the named class:

  % java HelloVoter ↩

  Notice the absence of `.class` suffix: JVM only needs the class name. If no run-time errors occur, glee of execution ensues...
- `javadoc` – to generate documentation using Doc-comments
- `javap` – to look inside the bytecode
- `jar` – to archive multiple (bytecode, docs and media) files into a single (executable) jar-file

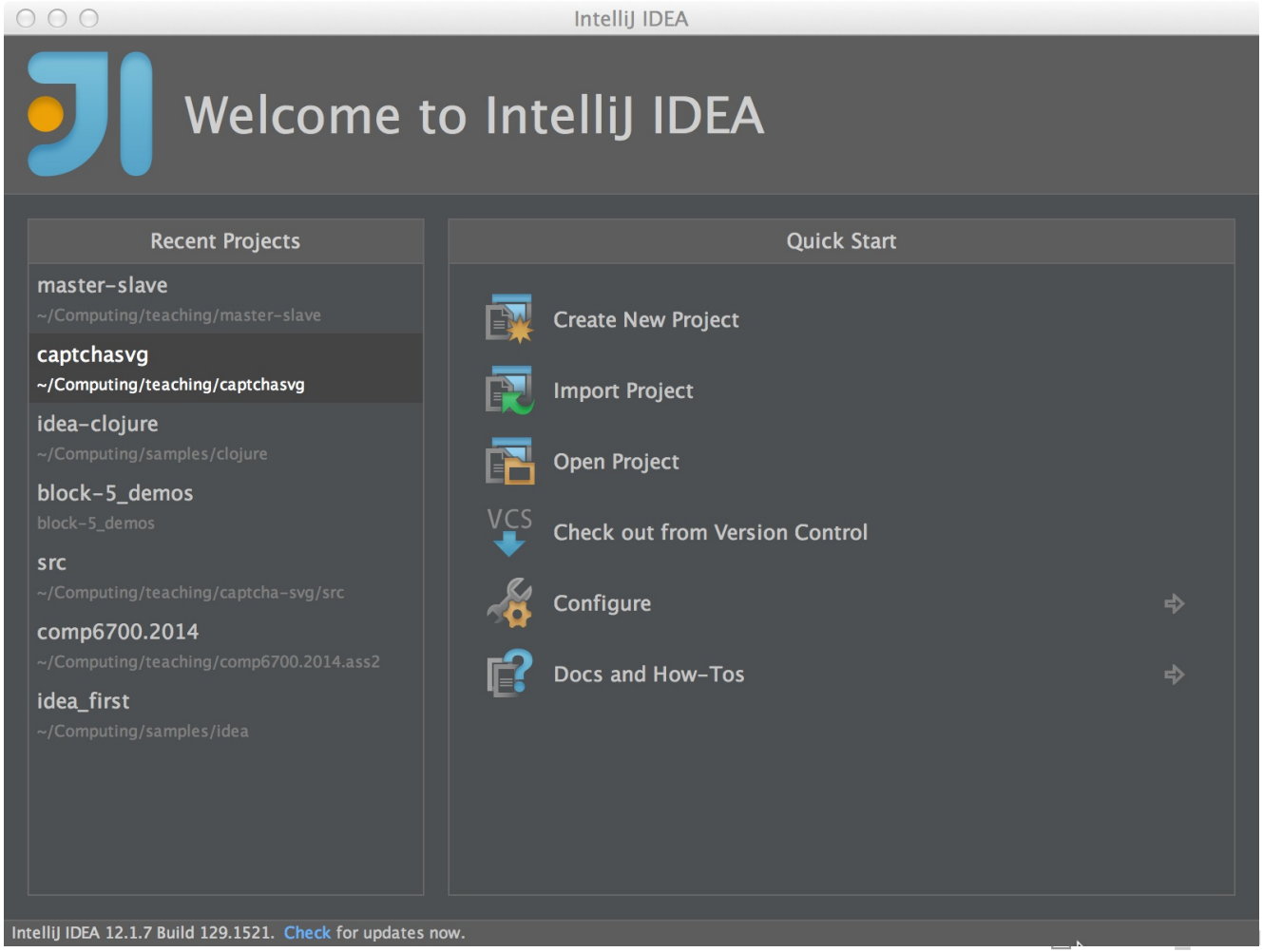
IDEs

- IntelliJ IDEA (CE)
- Eclipse (many distos, I din't bother to follow lately)
- Netbeans (from *Oracle*, like Java SE itself)

IntelliJ IDEA

Still not sure why is such name? But the application is superb!

In an IDE, everything requires a **project**:



IntelliJ IDEA

After few key strokes:

HelloVoter.java – code – [~/Computing/teaching/comp6700/master/local_docs/lectures/lecturettes/code]

```
/**
 * Execution of Dave Lister for stealing apple of the people
 */

import java.util.Scanner;

class HelloVoter {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("What is your name, voter? ");
        String name = sc.nextLine();
        System.out.
            printf("Kneel, voter %s, you are under the sentence of death!%n", name);
    }

}
```