Solutions

Each answer shows the steps, then the **result**.

2.1

- bytecode; virtual machine (or JVM); interprets

2.2

- breakpoint **D**; context-sensitive prompt **A**
- dynamic syntax check **B**; expand and collapse code blocks **C**
- prettyprint **C**; report window **D**
- single stepping **D**; variables and expressions window **D**

2.3

(a) context-sensitive prompt (auto-complete)

(b) dynamic syntax check

(c) expand and collapse code blocks

2.4

- the program pauses at that line, so the programmer can inspect the variables (and then step or continue)

2.5

- the code is displayed with keywords, identifiers, strings and comments in different colours or fonts
- and with consistent indentation, so the structure of the program is visible at a glance

2.6 any **two** of:

- the same bytecode runs on any computer that has a virtual machine
- the program does not have to be compiled again for each type of processor or operating system
- syntax errors are still found by the compiler before the program runs

3.1

(a) the Java source code is compiled once into bytecode

- the bytecode is the same file for every device; each device has its own virtual machine (JVM)
- the virtual machine interprets the bytecode into the machine code of that device as the app runs

(b) benefit: one compiled file runs on every device, so the app does not have to be compiled separately for each processor and operating system

- drawback: interpreting bytecode is slower than running machine code directly, so the app runs more slowly (unless the JVM uses just-in-time compilation)

3.2

(a) coding; as the programmer types, the IDE suggests the identifiers, keywords or parameters that fit at that point, so names are typed correctly and quickly

(b) presentation; the body of a loop, an IF or a subroutine can be hidden, leaving its header, so the programmer sees the outline of a long program and can find a section quickly

(c) debugging; it lists the errors, warnings and the output of the program in one place, so the programmer sees what went wrong and where

(d) initial error detection; the syntax is checked as the code is typed and a mistake is underlined at once, so it is corrected before the program is translated

3.3

- set a **breakpoint** at the start of `Average()`, so the program pauses when the function is called
- **single-step** through the function one statement at a time
- after each step, read the values of the variables (the total, the count) in the variables window and compare them with the expected values
- the first statement after which a value is wrong, for example the total being added to twice or the count being one too small, is the line with the error

3.4 two features, each described:

- **prettyprint**: keywords, identifiers and comments are shown in different colours, with consistent indentation, so the programmer can see the structure of each part at a glance
- **expand and collapse code blocks**: the bodies of subroutines and loops can be hidden, so the programmer reads the outline of the 2000 lines first and opens one part at a time

3.5 any **two**, each explained:

- it detects errors as the code is typed (dynamic syntax checks) and offers context-sensitive prompts, so many mistakes are avoided before compiling
- it contains a debugger with breakpoints, single stepping and a variables window, so a logic error can be found by watching the program run
- it presents the code with prettyprint and collapsible blocks, and reports errors and output in a report window, so the whole edit, run and fix cycle happens in one place