

5.2 Language Translators

Name: _____ Class: _____ Date: _____ Total: 30 marks

KEY WORDS interpreter 解释器 • compiler 编译器
• integrated development environment (ide) 集成开发环境 • assembler 汇编器
• assembly language 汇编语言

Objective

Build the skills to answer exam questions on **language translators** 翻译器 — assemblers, compilers and interpreters, when to use each, and the features of an IDE.

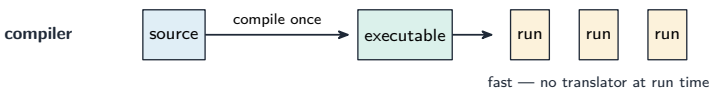
You must be able to:

- explain the need for an **assembler**, a **compiler** 编译器 and an **interpreter** 解释器
- compare and justify the use of a compiler or an interpreter
- explain the **hybrid** (bytecode) approach used by Java
- describe features of an **Integrated Development Environment (IDE)** 集成开发环境

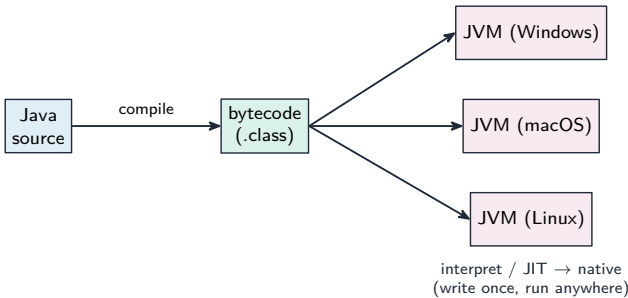
1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

Three translators



Compiler: translate all at once, then run. Interpreter: translate and run line by line.



Java compiles to bytecode run by a virtual machine

You write source code; the CPU runs **machine code** 机器码. A translator converts between them.

- **assembler** — translates **assembly language** to machine code (one-to-one).
- **compiler** — translates a whole **high-level** program to machine code **once, before it runs**.
- **interpreter** — translates and runs a high-level program **one line at a time**.

Compiler vs interpreter

	compiler	interpreter
When translated	all at once, before running	line by line, while running
Errors	reports all at compile time	stops at the first error reached
Output	a stand-alone executable	none — re-translates each run
Needs the translator to run?	no	yes
Run-time speed	faster	slower
Portable across platforms?	recompile per platform	yes (if interpreter exists)

Choosing: use a **compiler** for speed and to distribute finished software; use an **interpreter** for quick development and cross-platform scripts.

Java (hybrid) and IDEs

Java is **compiled to bytecode** 字节码, which a **virtual machine** (the JVM) interprets (or just-in-time compiles). So errors are caught early **and** the bytecode runs anywhere with a JVM (“write once, run anywhere”). An **IDE** brings the tools together, with features that help at each stage:

- **writing code:** an editor with **syntax highlighting**, **auto-complete** / context-sensitive prompts, and **prettyprinting** (auto-indenting).
- **finding errors: error diagnostics** that point to the line of a syntax error, and reports as you type.
- **debugging:** a **debugger** with **breakpoints**, **single-stepping**, and a **watch window** to inspect variable values while paused.

2 Practice

Now apply the methods above.

2.1 State the difference between a **compiler** and an **interpreter**. [2]

2.2 Give one situation in which an **interpreter** is more suitable than a compiler. [1]

2.3 State what an **assembler** translates. [1]

2.4 Name **two** features of an IDE. [2]

3 Exam-style questions

3.1 (a) State **two** benefits of using a **compiler** rather than an interpreter. [2]

(b) State **two** benefits of using an **interpreter** rather than a compiler. [2]

3.2 Explain how Java’s “compile to bytecode, then run on a virtual machine” approach lets the same program run on different computers. [3]

3.3 Describe **two** features an IDE provides to help a programmer **debug** a program. [2]

3.4 A company sells finished software to customers who do not have any programming tools installed. State which translator the company should use, and justify your choice. [2]

3.5 A programmer writes a program in a high-level language and releases it as **free software**.

(a) **Explain** how the programmer can use an **interpreter** first and then a **compiler** while developing the same program. [4]

(b) **State** two things an **Integrated Development Environment** provides beyond a translator. [2]

(c) A user downloads the program. **Describe** what “free software” allows the user to do that **freeware** does not. [3]

(d) **Explain** why the program still runs more slowly when interpreted than when compiled, even on the same computer. [3]

(e) The compiler reports a **syntax error** on a line that looks correct. **Suggest** one reason this can happen. [1]

Solutions

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

2.1

- a compiler translates the **whole** program **once** into an executable
- an interpreter translates and runs it **one line at a time**

2.2

- e.g. during development (quick edit–run cycles) / for a cross-platform script / a small or run-once program

2.3

- **assembly language** (into machine code)

2.4

- any two of: syntax highlighting; auto-complete; one-key compile/run; debugger; version-control integration

3.1

(a) any two: runs faster (no translation at run time); produces a stand-alone executable; the translator is not needed to run it; the source code is not given away

(b) any two: reports errors as it reaches them (easier development); no need to recompile after each change; the same program runs on any platform with the interpreter

3.2

- the program is compiled once into **bytecode**, which is platform-independent
- any computer with a **JVM** can interpret/run that bytecode
- so it runs unchanged on different operating systems — “write once, run anywhere”

3.3

- any two: set **breakpoints** to pause execution; **step through** line by line; **inspect variable** values while paused

3.4

- a **compiler**
- it produces a stand-alone executable that runs without any programming tools installed on the customer’s computer

3.5

(a) during development an **interpreter** translates and runs one statement at a time, so the programmer sees the effect of a change immediately and can stop at the first error

- once the program works, a **compiler** translates the whole source into machine code once
- that produces an executable that runs faster and can be distributed without the source

(b) any two: an editor with syntax highlighting and auto-complete; a debugger with breakpoints, single-stepping and variable watches; a project manager for multiple source files; built-in help and error reporting

(c) free software gives the user the source code and the right to **study**, **modify** and **redistribute** it, including modified versions

- freeware costs nothing to use but the source is not provided and the licence forbids changing or redistributing it

(d) an interpreter must **translate each statement every time it is executed**, so a statement inside a loop is translated on every pass

- compiled code was translated once, before running, so the processor executes machine code directly with no translation overhead at all

(e) the real error is on an **earlier** line — for example a missing closing bracket or quotation mark — and the compiler only notices when the statement fails to make sense