A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

5.2 Language Translators

Name: _____ Class: _____ Date: _____ **Total: 15 marks**

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.

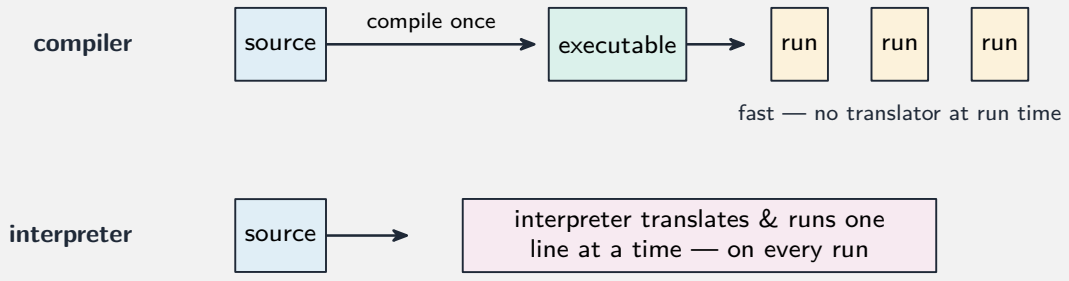
You must be able to:

- 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

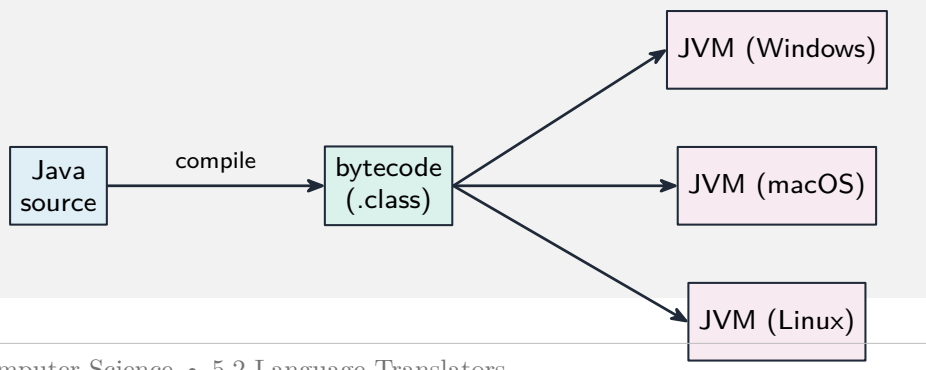
1 Worked examples

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.



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]

4 Go further

You are now ready for the real exam questions on this subtopic. Open the **5.2 Language Translators** past-paper sheet in the Library, or try these in **Practice mode**:

- 9618/11 N24 —Q4 (IDE features and translators)
- 9618/12 N25 —Q10 (compiler vs interpreter)
- 9618/13 J25 —Q3 (translators and bytecode)
- 9618/11 J24 —Q3 (language translators)

Solutions

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.