System Software

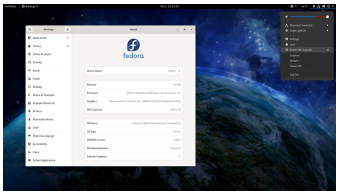
AS Computer Science ▪ Topic 5

A-Level Computer Science

applications
operating system
hardware

What we will cover

1. Operating systems
2. Utility software
3. Program libraries
4. Language translators
5. IDEs
6. Recap



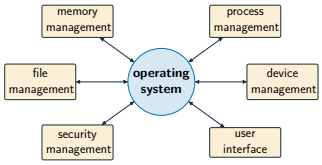
1 Operating systems

Why a computer needs an OS

The **operating system** 操作系统 is the software layer that:

- manages the hardware for programs
- provides services & a user interface
- lets programs share the hardware safely

What the OS manages

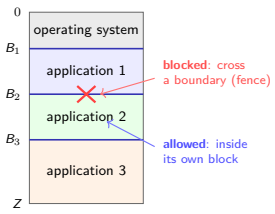


- **process management** 进程管理 – load, schedule and switch programs
- **memory management** 内存管理 – hand out memory and keep processes apart
- file management – folders and permissions
- **device management** 设备管理 – I/O and peripherals
- security, the user interface, networking

What each management task actually does

process	a running program is a **process** 进程: load it, schedule it on the CPU, switch between processes, kill the misbehaving
memory	give each process memory and keep them apart; **virtual memory** 虚拟内存 and **paging** 分页 let the working set exceed the physical RAM
file	organise files and folders on **secondary storage** 辅助存储器, control permissions, prevent write corruption
device	handle I/O through **device drivers** 设备驱动, buffer data, manage interrupts, and give every peripheral a uniform interface

Memory protection and utilities



each app kept in its own block

Utility programs 实用程序 maintain the system:

- **antivirus** 杀毒软件, **firewall** 防火墙
- **backup** 备份, **compression** 压缩
- **disk defragmenter** 碎片整理 (HDD only)

Which utility would you use?

an old hard disk reads files slowly – pieces scattered everywhere

→ **disk defragmenter**

an attachment is too big to email

→ **compression**

one careless delete could lose the whole project

→ **backup**

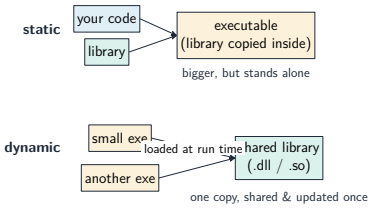
a borrowed USB stick might carry malware

→ **virus checker**

An exam answer names the utility and links it to the scenario's need.

2 Libraries

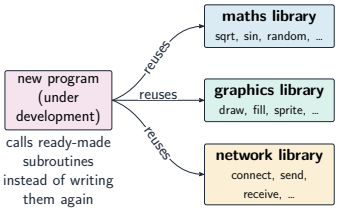
Program libraries



A **program library** 程序库 is reusable, tested code.

- **static library** 静态库: copied into the executable
- **dynamic library** 动态库 (`.dll`/`.so`): loaded at run time, shared

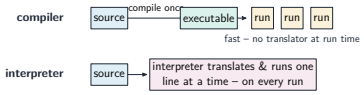
A library is code someone already tested



> Reusing ready-made routines saves time, is reliable (well-tested, widely used) and is standardised across every program that calls it.

3 Translators

Compiler vs interpreter



- **compiler** 编译器: translates once into a standalone **executable** 可执行文件. Faster at run time, but tied to one CPU and OS – recompile for each platform.
- **interpreter** 解释器: line by line, every run. It must be installed to run the program and is generally slower, but the same source ports anywhere.

The words this topic is built from

machine code 机器码	the binary instructions the **processor** 处理器 actually executes
assembly language 汇编语言	one mnemonic per machine instruction – readable, and still machine-specific
a high-level language	machine-independent, with **subroutines** 子程序, data structures and libraries
a compiler	translates the whole program once; errors are reported before it runs, and it runs fast
an interpreter	translates and runs line by line; errors appear as they are met, and it runs slower
which to use	an interpreter while developing, a compiler for the released product

> Compilation moves the work before running; interpretation moves it into running. Everything else about the comparison follows from that.

Choosing between them

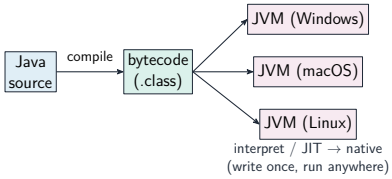
Use a compiler when	Use an interpreter when
run-time speed matters	you want fast edit–run cycles
distributing to users without dev tools	writing cross-platform scripts
the program runs many times	the program is small or run once
	teaching beginners

> Neither is "better". The question is always how often the program runs against how often you change it.

Hybrid: Java bytecode

Java compiles to **bytecode** 字节码, run by a **virtual machine** 虚拟机 (JVM).

> Why both? A compiler alone ties the program to one platform; an interpreter alone is slow. Compiling to **bytecode** 字节码 does the heavy translation once and catches errors early; the JVM then runs it anywhere, with **JIT** 即时编译 near native speed.



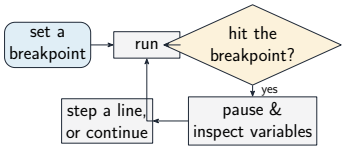
Java's hybrid model, and what an IDE gives you

bytecode 字节码	Java is compiled once into a platform-independent intermediate form
the virtual machine 虚拟机	the JVM then interprets it – so one compiled file runs anywhere a JVM exists
just-in-time compilation 即时编译	the JVM compiles hot bytecode to native machine code while running, recovering most of the speed
syntax highlighting 语法高亮	keywords, strings and comments in different colours, plus auto-indent and bracket matching
auto-complete 自动补全	suggests names and shows a function's parameters as you type
and the rest	**version control** 版本控制 integration, project management, a debugger, **refactoring** 重构 tools, and **unit test** 单元测试 runners

> An IDE speeds development by putting writing → running → debugging → fixing behind one interface. Common ones: Visual Studio, PyCharm, Eclipse.

4 IDE

Integrated Development Environment

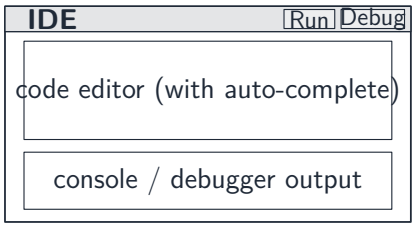


An **IDE** 集成开发环境 bundles the dev tools:

- editor with **syntax highlighting** 语法高亮, **auto-complete** 自动补全
- one-key compile/run; inline errors
- **debugger** 调试器: **breakpoints** 断点, step, inspect

An IDE, on screen

Editor, Run button and debugger in one window – that is the whole idea.



Utility software

Antivirus; **backup** 备份 software that copies user data elsewhere; compression; a **defragmenter** 碎片整理程序; disk analysis and repair for file-system errors and bad sectors; and a **firewall** 防火墙 filtering traffic by rules. Bundling them with the OS saves the user installing each one.

utility programs

antivirus | backup | file compression | disk defragmenter

Worked example – why compile to bytecode at all?

Java is compiled to bytecode, which a JVM interprets. Why both, instead of compiling straight to machine code?

- A compiler produces machine code for *one* processor and OS, so the program will not run elsewhere.
- Java targets a **virtual** machine, so the bytecode is identical everywhere and each platform supplies its own JVM – "write once, run anywhere".
- The price is speed, which is why a real JVM compiles hot paths just in time.

Portability against speed. Naming the trade-off, and the JIT that softens it, is what a full answer needs.

Exam tips

- List the OS's jobs (memory, process, file, device and security management) – "manages resources" alone is too vague.
- Compare **compiler vs interpreter vs assembler**: what each translates and when errors are reported.
- Explain what an **IDE** provides (editor, debugger, error diagnostics, auto-complete).

5 Recap

Topic 5 – key takeaways

1 OS

manages CPU, memory, files, devices, security

2 Libraries

reusable code; static vs dynamic

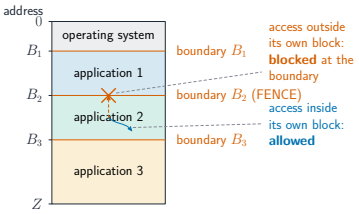
3 Translators

compiler once vs interpreter line-by-line

4 IDE

editor, translator & debugger in one

Memory protection keeps processes apart



Memory protection keeps each application in its own block of memory

Check yourself

1. Which OS job keeps each program in its own memory?
2. Compiler or interpreter: which makes a standalone file?
3. Why is Java bytecode portable?
4. Name a debugger feature in an IDE.

apps
OS
hardware

Answers **1.** memory management (protection) **2.** compiler **3.** any JVM runs it **4.** breakpoints / step / inspect