TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN
& TRUYỀN THÔNG

Giáo trình

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

Căn bản & Nâng cao



Phạm Thị Thương · Tô Hữu Nguyên
Nguyễn Văn Núi · Trần Hải Thanh

MỤC LỤC

Danh mục từ viết tắt vii

Lời nói đầu ix

1 Tổng quan 1

1.1 Giới thiệu phương pháp tiếp cận trong lập trình . . . 1
1.1.1 Lập trình tuyến tính . . . 1
1.1.2 Lập trình hướng chức năng . . . 2
1.1.3 Lập trình hướng đối tượng . . . 3
1.2 Các khái niệm cơ bản của lập trình hướng đối tượng . . . 4
1.2.1 Đối tượng . . . 4
1.2.2 Lớp đối tượng . . . 5
1.2.3 Bao bọc và che giấu thông tin . . . 5
1.2.4 Kế thừa . . . 5
1.2.5 Đa hình . . . 6
1.3 Cách tiếp cận và giải quyết bài toán theo phương pháp hướng đối tượng . 6
1.3.1 Mô tả bài toán . . . 7
1.3.2 Đặc tả yêu cầu . . . 7
1.3.3 Trích chọn đối tượng . . . 8
1.3.4 Mô hình hoá lớp đối tượng . . . 9
1.3.5 Thiết kế tổng quát . . . 10
1.3.6 Thiết kế chi tiết . . . 10
1.4 Một số ngôn ngữ lập trình hướng đối tượng . . . 11
1.5 Giới thiệu một số Editor biên soạn chương trình Java . . . 14
1.5.1 Eclipse . . . 14
1.5.2 NetBeans . . . 14
1.5.3 Một số IDE thông dụng khác . . . 15
1.6 Tổng kết chương . . . 17
Ôn tập cuối chương . . . 17

2 Giới thiệu ngôn ngữ lập trình Java 20

2.1 Giới thiệu chung . . . 20
2.1.1 Lịch sử hình thành và phát triển của Java . . . 20
2.1.2 Ngữ cảnh các ứng dụng của ngôn ngữ Java . . . 21
2.2 Môi trường lập trình Java . . . 22
2.2.1 Máy ảo Java (JVM — Java Virtual Machine) . . . 23
2.2.2 Bộ công cụ phát triển JDK . . . 25
2.2.3 Java Runtime Environment (JRE) . . . 25
2.2.4 Java Development Environment (IDE) . . . 26
2.3 Cấu trúc của một chương trình Java . . . 26
2.4 Biên dịch và thông dịch chương trình Java . . . 30
2.5 Các dạng chương trình ứng dụng của Java . . . 33
2.5.1 Phân loại theo cách vận hành, dịch và chạy chương trình . . . 33
2.5.2 Phân loại theo dạng chương trình ứng dụng, mục đích sử dụng . . . 34
2.6 Case Study . . . 36
2.7 Ví dụ áp dụng . . . 42
2.8 Tổng kết chương . . . 47

3.1.1 Định danh . . . 50
3.1.2 Từ khoá . . . 51
3.1.3 Toán tử . . . 52
3.1.4 Chú thích . . . 53
3.2 Các kiểu dữ liệu trong Java . . . 54
3.2.1 Kiểu nguyên thuỷ . . . 55
3.2.2 Kiểu tham chiếu . . . 57
3.2.3 Ép kiểu . . . 58
3.2.4 Lớp bao bọc . . . 59
3.3 Biến, hằng, biểu thức và các phép toán cơ bản . . . 60
3.3.1 Biến . . . 60
3.3.2 Hằng . . . 62
3.3.3 Biểu thức . . . 62
3.4 Các câu lệnh nhập xuất dữ liệu cơ bản qua các thiết bị ngoại vi . . . 63
3.4.1 Nhập dữ liệu từ bàn phím . . . 63
3.4.2 Xuất dữ liệu ra màn hình . . . 66
3.5 Các cấu trúc điều khiển chương trình trong Java . . . 69
3.5.1 Câu lệnh rẽ nhánh . . . 69
3.5.2 Các câu lệnh lặp . . . 72
3.5.3 Các lệnh chuyển vị . . . 75
3.6 Cấu trúc mảng, xâu ký tự trong Java . . . 77
3.6.1 Cấu trúc mảng . . . 77
3.6.2 Xâu ký tự . . . 80
3.7 Case Study . . . 85
3.8 Ví dụ áp dụng . . . 89
3.9 Tổng kết chương . . . 92
3.10 Bài tập thực hành . . . 92

4 Ngoại lệ và xử lý ngoại lệ trong Java 95
4.1 Cấu trúc phân cấp của các lớp xử lý ngoại lệ . . . 95
4.1.1 Cây phân cấp Throwable . . . 95
4.1.2 Error — Lỗi nghiêm trọng . . . 95
4.1.3 Exception — Khi "có chuyện không mong đợi" . . . 96
4.1.4 Ví dụ thực tế: Phân loại ngoại lệ trong chương trình quản lý điểm 98
4.1.5 Phân biệt Error, Checked Exception, và Unchecked Exception — Nhìn từ góc độ trách nhiệm . . . 98
4.2 Khối lệnh try, catch và finally . . . 99
4.2.1 Cú pháp cơ bản: try-catch . . . 99
4.2.2 Bắt nhiều loại ngoại lệ: Nhiều khối catch . . . 100
4.2.3 Khối finally — Dọn dẹp trong mọi tình huống . . . 102
4.2.4 try-with-resources — Cách hiện đại để quản lý tài nguyên (Java 7+) 103
4.2.5 Tổng hợp: Quyết định dùng try-catch khi nào . . . 104
4.3 Ném ngoại lệ bằng lệnh `throw` . . . 105
4.3.1 Cú pháp và cơ chế của throw . . . 105
4.3.2 Khi nào cần chủ động throw? . . . 106
4.3.3 Ví dụ 1: Kiểm tra tham số đầu vào — Validation . . . 106

4.3.4 Ví dụ 2: Phân biệt throw với try-catch — Ai chịu trách nhiệm? . . 108
4.3.5 Ví dụ 3: Re-throw — Bắt rồi ném tiếp 109
4.3.6 Chọn loại ngoại lệ phù hợp để throw 110
4.4 Mệnh đề throws 111
4.4.1 Cú pháp và cơ chế của throws 111
4.4.2 throw và throws — Hai khái niệm, hai vai trò 112
4.4.3 Checked Exception và sự bắt buộc của throws 113
4.4.4 throws và nguyên tắc thiết kế — Ai nên xử lý? 114
4.4.5 throws cho unchecked exception — Có nên không? 114
4.5 Tự định nghĩa ngoại lệ 115
4.5.1 Cách tạo lớp ngoại lệ tuỳ chỉnh 115
4.5.2 Checked hay Unchecked — Quyết định kế thừa từ đâu 117
4.5.3 Ví dụ thực tế: Hai ngoại lệ tuỳ chỉnh cho hệ thống đặt hàng 117
4.5.4 Quy ước và thực hành tốt khi tạo ngoại lệ tuỳ chỉnh 119
4.6 Lần vết ngoại lệ StackTrace 121
4.6.1 Cấu trúc của một StackTrace 121
4.6.2 Cách đọc StackTrace để tìm nguyên nhân gốc rễ 122
4.6.3 Chuỗi nguyên nhân — Caused by và Suppressed 124
4.6.4 Quy ước và mẹo đọc StackTrace nhanh 125
4.7 Case Study 126
4.7.1 Bài toán 126
4.7.2 Thiết kế ngoại lệ 126
4.7.3 Cài đặt 126
4.8 Ví dụ áp dụng 132
4.8.1 Ví dụ 1: Đọc file an toàn với try-catch-finally 132
4.8.2 Ví dụ 2: Chuỗi trách nhiệm với throw và throws 134
4.8.3 Ví dụ 3: Bọc ngoại lệ — từ lỗi kỹ thuật đến lỗi nghiệp vụ 136
4.9 Tổng kết chương 138
4.10 Bài tập thực hành 139

5 Lớp đối tượng và các thành phần của lớp 142
5.1 Lớp đối tượng 142
5.1.1 Định nghĩa lớp đối tượng và các thành phần của lớp 142
5.1.2 Các thuộc tính dữ liệu của lớp 143
5.1.3 Định nghĩa hàm thành phần 145
5.1.4 Truyền tham số và lời gọi hàm 147
5.1.5 Toán tử tạo lập đối tượng 149
5.2 Các thuộc tính kiểm soát truy nhập các thành phần của lớp 153
5.2.1 Bốn mức kiểm soát truy nhập 153
5.2.2 Nguyên tắc thiết kế — Che giấu thông tin 156
5.2.3 Kiểm soát truy nhập ở cấp lớp 158
5.2.4 Bảng tham khảo nhanh — Chọn bổ từ nào? 158
5.2.5 Ví dụ tổng hợp — Lớp Student với kiểm soát truy nhập đầy đủ . . 159
5.3 Tạo và sử dụng đối tượng 161
5.3.1 Tạo đối tượng với toán tử new 161
5.3.2 Truy cập thành phần của đối tượng — Dấu chấm 163

5.3.3 Tham chiếu đối tượng và null . . . 164
5.3.4 Đối tượng trong bộ nhớ — Stack và Heap . . . 164
5.3.5 Garbage Collection — Dọn rác tự động . . . 166
5.3.6 Mảng đối tượng . . . 166
5.3.7 Ví dụ tổng hợp — Quản lý danh sách sinh viên . . . 168
5.4 Case Study . . . 170
5.4.1 Bài toán . . . 170
5.4.2 Thiết kế lớp . . . 170
5.4.3 Chương trình chính — Lớp QuanLyDiem . . . 175
5.5 Ví dụ áp dụng . . . 175
5.5.1 Ví dụ 1: Lớp PhanSo — Constructor và nạp chồng phương thức . 176
5.5.2 Ví dụ 2: Lớp MaTran — Mảng đối tượng và thao tác tập thể . . . 178
5.6 Tổng kết chương . . . 182
5.7 Bài tập thực hành . . . 183
6 Gói và các lớp cơ sở trong gói java.lang 185
6.1 Gói . . . 185
6.1.1 Tạo gói cho chương trình và cách sử dụng gói . . . 185
6.1.2 Giới thiệu một số gói quan trọng của Java . . . 189
6.2 Lớp Collection . . . 191
6.2.1 Tổng quan về Collections Framework . . . 191
6.2.2 ArrayList — Mảng động thay thế cho mảng tĩnh . . . 191
6.2.3 ArrayList và Mảng — Khi nào dùng gì? . . . 195
6.2.4 Chuyển đổi giữa ArrayList và Mảng . . . 195
6.2.5 Giới thiệu ngắn về HashMap . . . 196
6.3 Các lớp cơ sở trong gói java.lang . . . 198
6.3.1 String — Xử lý văn bản . . . 198
6.3.2 Math — Tính toán toán học . . . 198
6.3.3 Lớp Wrapper — “Bọc” kiểu nguyên thuỷ thành đối tượng . . . 199
6.3.4 System — Tương tác với hệ thống . . . 200
6.3.5 Object — Lớp gốc của mọi lớp . . . 201
6.3.6 Các lớp Wrapper nâng cao và tiện ích khác . . . 201
6.4 Case Study . . . 202
6.4.1 Bài toán . . . 202
6.4.2 Thiết kế kiến trúc package . . . 202
6.4.3 Cài đặt . . . 203
6.5 Ví dụ áp dụng . . . 207
6.5.1 Ví dụ 1: Tổ chức dự án với package và import . . . 207
6.5.2 Ví dụ 2: Quản lý danh sách công việc với ArrayList . . . 210
6.6 Tổng kết chương . . . 212
6.6.1 Các khái niệm cốt lõi . . . 213
6.6.2 Mối quan hệ giữa các khái niệm . . . 213
6.6.3 Bảng đối chiếu nhanh . . . 213
6.6.4 Những điều cần nhớ . . . 213
6.7 Bài tập thực hành . . . 213
7 Kế thừa 216

7.1 Quan hệ kế thừa . . . 216
7.1.1 Khái niệm kế thừa . . . 216
7.1.2 Cơ chế kế thừa trong Java . . . 218
7.1.3 Ví dụ từng bước — Từ DongVat đến Heo . . . 219
7.1.4 Kiểm tra kiểu với instanceof . . . 221
7.2 Thiết kế cây thừa kế . . . 222
7.2.1 Nguyên tắc thiết kế cây thừa kế . . . 222
7.2.2 Xây dựng cây thừa kế từng bước . . . 224
7.2.3 Những lỗi thiết kế thường gặp . . . 228
7.3 Viết đè . . . 229
7.3.1 Cú pháp và annotation @Override . . . 229
7.3.2 Ghi đè và gọi phương thức gốc với super . . . 231
7.3.3 Ghi đè (Override) và Nạp chồng (Overload) — Đừng nhầm lẫn . . . 231
7.3.4 Ghi đè toString() và equals() — Kế thừa từ Object . . . 233
7.4 Quan hệ is-a và has-a . . . 234
7.4.1 IS-A — Quan hệ kế thừa . . . 234
7.4.2 HAS-A — Quan hệ composition (chứa trong) . . . 235
7.4.3 So sánh IS-A và HAS-A — Khi nào dùng gì? . . . 236
7.4.4 Ví dụ đối chiếu — Cùng một bài toán, hai cách thiết kế . . . 237
7.4.5 Composition over Inheritance — Nguyên tắc thiết kế quan trọng . . . 239
7.5 Case Study . . . 240
7.5.1 Bài toán . . . 240
7.5.2 Thiết kế cây thừa kế . . . 240
7.5.3 Cài đặt . . . 241
7.5.4 Chương trình chính — Quản lý lương . . . 244
7.6 Ví dụ áp dụng . . . 245
7.6.1 Ví dụ 1: Kế thừa và constructor chaining . . . 245
7.6.2 Ví dụ 2: Phân biệt Override và Overload trong thực tế . . . 247
7.7 Tổng kết chương . . . 248
7.7.1 Các khái niệm cốt lõi . . . 248
7.7.2 Mối quan hệ giữa các khái niệm . . . 248
7.7.3 Bảng đối chiếu nhanh . . . 249
7.7.4 Những điều cần nhớ . . . 249
7.8 Bài tập thực hành . . . 250

8 Đa hình và lớp trừu tượng 252

8.1 Đa hình . . . 252
8.1.1 Đặc điểm của đa hình . . . 252
8.1.2 Đa hình động . . . 254
8.1.3 Đa hình tĩnh . . . 260
8.2 Lớp trừu tượng . . . 262
8.2.1 Khái niệm . . . 263
8.2.2 Lớp trừu tượng và lớp cụ thể . . . 265
8.2.3 Phương thức trừu tượng . . . 268
8.2.4 Lớp Object . . . 273
8.3 Case Study . . . 277

8.3.1 Bài toán . . . 277
8.3.2 Thiết kế cây thừa kế . . . 277
8.3.3 Cài đặt . . . 278
8.3.4 Chương trình chính — Quản lý đơn hàng . . . 281
8.4 Ví dụ áp dụng . . . 282
8.4.1 Ví dụ 1: Đa hình động với upcasting . . . 282
8.4.2 Ví dụ 2: Nạp chồng phương thức và constructor . . . 285
8.5 Tổng kết chương . . . 287
8.6 Bài tập thực hành . . . 287

9 Giao diện và các lớp trong 290
9.1 Giao diện (interfaces) . . . 290
9.1.1 Định nghĩa . . . 290
9.1.2 Tính chất . . . 294
9.1.3 Sử dụng . . . 298
9.2 Lớp trong (inner classes) . . . 304
9.2.1 Định nghĩa . . . 304
9.2.2 Tính chất . . . 306
9.2.3 Phân loại . . . 309
9.2.4 Sử dụng . . . 313
9.3 Case Study . . . 318
9.3.1 Bài toán . . . 319
9.3.2 Thiết kế với Interface . . . 319
9.3.3 Cài đặt . . . 319
9.3.4 Chương trình chính — Trung tâm điều khiển . . . 326
9.4 Ví dụ áp dụng . . . 326
9.4.1 Ví dụ 1: Interface và đa kế thừa kiểu . . . 326
9.5 Tổng kết chương . . . 330
9.6 Bài tập thực hành . . . 330

Kết luận và định hướng mở rộng 334

Tài liệu tham khảo 339

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Cụm từ đầy đủ	Diễn giải ngắn
API	Application Programming Interface	Giao diện lập trình ứng dụng
ART	Android Runtime	Môi trường thực thi ứng dụng Android
CLDC	Connected Limited Device Configuration	Cấu hình thiết bị kết nối giới hạn (Java ME)
DEX	Dalvik Executable	Định dạng mã thực thi trên Android Runtime
DIP	Dependency Inversion Principle	Nguyên lý đảo ngược phụ thuộc (D trong SOLID)
DJL	Deep Java Library	Thư viện học sâu mã nguồn mở cho Java
DRY	Don’t Repeat Yourself	Nguyên tắc không lặp lại mã nguồn
EJB	Enterprise JavaBeans	Thành phần nghiệp vụ phía máy chủ trong Java EE
ERP	Enterprise Resource Planning	Hệ thống hoạch định nguồn lực doanh nghiệp
GC	Garbage Collector	Cơ chế thu gom bộ nhớ tự động của JVM
GUI	Graphical User Interface	Giao diện đồ họa người dùng
HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản
HTTP	HyperText Transfer Protocol	Giao thức truyền siêu văn bản
IDE	Integrated Development Environment	Môi trường phát triển tích hợp
ISP	Interface Segregation Principle	Nguyên lý phân tách giao diện (I trong SOLID)
JAR	Java Archive	Định dạng gói nén cho ứng dụng Java
JDBC	Java Database Connectivity	API kết nối cơ sở dữ liệu trong Java
JDK	Java Development Kit	Bộ công cụ phát triển Java
JIT	Just-In-Time (Compiler)	Trình biên dịch tức thời trong JVM
JMS	Java Message Service	Dịch vụ thông điệp bất đồng bộ trong Java EE
JPA	Java Persistence API	API ánh xạ đối tượng–quan hệ trong Java
JRE	Java Runtime Environment	Môi trường chạy Java
JSON	JavaScript Object Notation	Định dạng trao đổi dữ liệu nhẹ
JSP	JavaServer Pages	Công nghệ tạo trang web động phía máy chủ
JVM	Java Virtual Machine	Máy ảo Java
KISS	Keep It Simple, Stupid	Nguyên tắc thiết kế đơn giản
LCMS	Learning Content Management System	Hệ thống quản lý nội dung học tập
LSP	Liskov Substitution Principle	Nguyên lý thay thế Liskov (L trong SOLID)
LTS	Long-Term Support	Phiên bản hỗ trợ dài hạn (Java)
MIDP	Mobile Information Device Profile	Hồ sơ thiết bị thông tin di động (Java ME)
MVC	Model-View-Controller	Mẫu kiến trúc Model–View–Controller
NFTC	Oracle No-Fee Terms and Conditions	Điều khoản miễn phí của Oracle cho JDK
OCP	Open/Closed Principle	Nguyên lý đóng–mở (O trong SOLID)

Từ viết tắt	Cụm từ đầy đủ	Diễn giải ngắn
OOP	Object-Oriented Programming	Lập trình hướng đối tượng
ORM	Object-Relational Mapping	Ánh xạ đối tượng–quan hệ
RAG	Retrieval-Augmented Generation	Kỹ thuật tăng cường truy xuất trong AI
REST	Representational State Transfer	Phong cách kiến trúc cho Web API
SDK	Software Development Kit	Bộ công cụ phát triển phần mềm
SOLID	Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion	Năm nguyên lý thiết kế hướng đối tượng
SQL	Structured Query Language	Ngôn ngữ truy vấn có cấu trúc
SRP	Single Responsibility Principle	Nguyên lý trách nhiệm đơn (S trong SOLID)
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất
URL	Uniform Resource Locator	Định vị tài nguyên thống nhất
WAR	Web Application Archive	Định dạng gói triển khai ứng dụng web Java
XML	eXtensible Markup Language	Ngôn ngữ đánh dấu mở rộng
YAGNI	You Ain’t Gonna Need It	Nguyên tắc không cài đặt trước những gì chưa cần

LỜI NÓI ĐẦU

Giáo trình Lập trình Hướng đối tượng với Java được biên soạn nhằm phục vụ học phần Lập trình hướng đối tượng thuộc chương trình đào tạo ngành Công nghệ Thông tin, Đại học Công nghệ Thông tin và Truyền thông Thái Nguyên (ICTU). Giáo trình trang bị cho sinh viên nền tảng vững chắc về tư duy hướng đối tượng và kỹ năng lập trình Java — một trong những ngôn ngữ phổ biến nhất trong công nghiệp phần mềm hiện đại.

Đối tượng sử dụng chính là sinh viên năm thứ hai ngành Công nghệ Thông tin, đã hoàn thành học phần Nhập môn lập trình với C/C++. Người học được kỳ vọng quen thuộc với cú pháp cơ bản, cấu trúc điều khiển và hàm, nhưng chưa có kiến thức hệ thống về lập trình hướng đối tượng. Giáo trình được thiết kế để người học có thể tự đọc, tự thực hành, phù hợp cả hình thức học trên lớp lẫn tự học từ xa.

Nội dung giáo trình trải qua chín chương, tổ chức theo mạch từ nền tảng đến chuyên sâu. Chương 1 đặt nền móng với ba phương pháp tiếp cận lập trình và sáu khái niệm trụ cột của OOP. Chương 2 giới thiệu ngôn ngữ Java, môi trường phát triển và quy trình biên dịch–thông dịch. Chương 3 trang bị cú pháp Java cốt lõi, tận dụng nền C/C++ đã có để tập trung vào điểm khác biệt. Chương 4 trình bày cơ chế xử lý ngoại lệ. Năm chương tiếp theo (5–9) tạo thành trục trung tâm, lần lượt xây dựng năng lực thiết kế hướng đối tượng qua lớp và đối tượng, gói và thư viện lõi, kế thừa, đa hình và lớp trừu tượng, giao diện và lớp trong. Xuyên suốt các chương này, hệ thống quản lý sinh viên được phát triển dần — từ lớp `SinhVien` đơn giản đến hệ thống phân cấp hoàn chỉnh với kế thừa, đa hình và giao diện — giúp người học thấy được sự tiến hóa của thiết kế qua từng khái niệm mới.

Về phương pháp, giáo trình tuân thủ triết lý "từ bài toán đến khái niệm" và "tiếp cận được nhưng không hạ chuẩn". Mỗi chương mở đầu bằng tình huống có ý nghĩa, dẫn dắt người học hình thành trực giác trước khi chuẩn hóa thành khái niệm và nguyên lý chính xác. Ví dụ được thiết kế theo nguyên tắc giàn giáo: làm mẫu cách suy nghĩ ở ví dụ đầu, giảm dần gợi ý để người học tự lực. Cuối mỗi section có câu hỏi tự kiểm tra; mỗi chương khép lại bằng tổng kết, trắc nghiệm và bài tập thực hành phân tầng từ nhận biết đến vận dụng.

Nhóm biên soạn chân thành cảm ơn các đồng nghiệp tại Khoa Công nghệ Thông tin, ICTU đã đóng góp ý kiến quý báu. Chúng tôi cũng trân trọng ghi nhận các tác giả của những tài liệu tham khảo được liệt kê ở cuối giáo trình. Mặc dù đã nỗ lực hết sức, giáo trình không tránh khỏi thiếu sót. Chúng tôi mong nhận được ý kiến đóng góp từ quý đồng nghiệp và bạn đọc để giáo trình được hoàn thiện hơn trong các lần tái bản sau.

Thái Nguyên, 2026

Phạm Thị Thương
Tô Hữu Nguyên
Nguyễn Văn Núi
Trần Hải Thanh

CHƯƠNG 1
TỔNG QUAN

Mục tiêu chương
Sau khi học xong chương này, người học có thể:

- Phân biệt ba phương pháp tiếp cận trong lập trình: tuyến tính, hướng chức năng, và hướng đối tượng.
- Trình bày sáu khái niệm cơ bản của OOP và vai trò của từng khái niệm.
- Mô tả quy trình sáu bước phân tích và thiết kế bài toán theo phương pháp hướng đối tượng.
- So sánh các ngôn ngữ OOP phổ biến và các IDE thông dụng cho lập trình Java.

1.1 Giới thiệu phương pháp tiếp cận trong lập trình

Khi giải quyết một bài toán bằng máy tính, người lập trình trước hết cần một **cách tư duy** — một **phương pháp tiếp cận** (programming paradigm) để tổ chức lời giải. Ba phương pháp chính đã lần lượt xuất hiện, mỗi phương pháp ra đời để giải quyết hạn chế của phương pháp trước đó: lập trình tuyến tính, lập trình hướng chức năng, và lập trình hướng đối tượng.

1.1.1 Lập trình tuyến tính

Lập trình tuyến tính (linear programming) là phương pháp tiếp cận sớm nhất và đơn giản nhất. Trong phương pháp này, chương trình được tổ chức như một **chuỗi tuần tự** các câu lệnh: lệnh thứ nhất thực hiện xong, đến lệnh thứ hai, rồi lệnh thứ ba, và cứ tiếp tục như vậy từ trên xuống dưới.

Hãy hình dung một chương trình nhỏ tính diện tích hình chữ nhật theo lối tuyến tính:

```
// Nhập chiều dài và chiều rộng
double dai = 5.0;
double rong = 3.0;
// Tính diện tích
double dienTich = dai * rong;
// In kết quả
System.out.println("Diện tích: " + dienTich);
```

Toàn bộ chương trình là một dãy lệnh chạy thẳng từ dòng đầu tiên đến dòng cuối cùng. Cách viết này rất trực quan và dễ hiểu đối với những chương trình nhỏ.

Tuy nhiên, khi bài toán trở nên phức tạp hơn — chẳng hạn cần tính diện tích cho nhiều hình khác nhau, hoặc cần xử lý nhiều trường hợp nhập liệu khác nhau — đoạn mã

tuyến tính nhanh chóng trở nên dài, lặp lại và khó bảo trì. Người lập trình bắt đầu tự hỏi: liệu có cách nào để **đóng gói** những đoạn lệnh thường dùng thành các khối riêng, có thể gọi lại nhiều lần không? Câu hỏi đó dẫn đến sự ra đời của phương pháp tiếp cận thứ hai.

> **Lưu ý:** Trong giáo trình này, thuật ngữ *lập trình tuyến tính* dùng để chỉ phương pháp tổ chức mã nguồn theo chuỗi tuần tự (còn gọi là lập trình thủ tục đơn giản). Không nên nhầm lẫn với *quy hoạch tuyến tính* (linear programming) trong toán học — một lĩnh vực hoàn toàn khác.

1.1.2 Lập trình hướng chức năng

Lập trình hướng chức năng (functional-oriented programming) giải quyết hạn chế của lập trình tuyến tính bằng cách tổ chức chương trình thành các **hàm** (function) — những khối mã độc lập, mỗi khối đảm nhận một nhiệm vụ cụ thể và có thể được gọi đi gọi lại nhiều lần từ bất kỳ đâu trong chương trình.

Quay lại ví dụ tính diện tích. Với cách tiếp cận hướng chức năng, ta có thể tách phần tính diện tích thành một hàm riêng:

```
// Hàm tính diện tích hình chữ nhật
double tinhDienTichHCN(double dai, double rong) {
    return dai * rong;
}

// Sử dụng hàm
double dt1 = tinhDienTichHCN(5.0, 3.0);
double dt2 = tinhDienTichHCN(7.0, 2.5);
System.out.println("Diện tích 1: " + dt1);
System.out.println("Diện tích 2: " + dt2);
```

Hàm `tinhDienTichHCN` đóng gói công thức tính diện tích. Mỗi khi cần tính, ta chỉ việc gọi hàm với tham số tương ứng, không phải viết lại công thức. Cách tổ chức này mang lại ba lợi ích quan trọng:

1. **Tái sử dụng mã nguồn:** Cùng một hàm có thể được gọi nhiều lần với dữ liệu khác nhau, giảm trùng lặp mã.
2. **Dễ bảo trì:** Khi cần sửa cách tính, ta chỉ cần sửa *một chỗ* duy nhất — bên trong thân hàm.
3. **Phân chia trách nhiệm:** Chương trình lớn được chia thành các hàm nhỏ, mỗi hàm giải quyết một phần việc rõ ràng, giúp nhiều người có thể cùng làm việc trên các phần khác nhau.

Tuy nhiên, lập trình hướng chức năng cũng bộc lộ giới hạn khi quy mô chương trình tiếp tục tăng. Hãy thử nghĩ về một hệ thống quản lý sinh viên: ta không chỉ có vài hàm tính toán đơn lẻ, mà còn có hàng chục thực thể như *Sinh viên*, *Lớp học*, *Giảng viên*, *Môn học*, mỗi thực thể mang theo dữ liệu riêng (tên, mã số, điểm) *và* các thao tác gắn liền với dữ liệu đó (đăng ký môn, tính điểm trung bình, xếp loại). Trong mô hình hướng chức năng, dữ liệu và hàm xử lý dữ liệu nằm tách rời nhau. Khi hệ thống phát triển, việc

quản lý mối liên kết giữa hàng trăm hàm và hàng trăm trường dữ liệu trở nên rất phức tạp và dễ sinh lỗi.

Nhu cầu **gắn kết dữ liệu và hành vi thành một đơn vị thống nhất** chính là động lực cho sự ra đời của phương pháp tiếp cận thứ ba.

1.1.3 Lập trình hướng đối tượng

Lập trình hướng đối tượng (object-oriented programming — OOP) giải quyết vấn đề trên bằng một ý tưởng cốt lõi: thay vì tách rời dữ liệu và hàm, hãy **đóng gói** chúng lại với nhau thành các **đối tượng** (object). Một đối tượng trong chương trình phản ánh trực tiếp một thực thể trong thế giới thực: một sinh viên, một lớp học, một chiếc xe — mỗi đối tượng vừa *biết* thông tin về chính nó (dữ liệu), vừa *biết cách* thực hiện các hành vi liên quan (hàm).

Hãy so sánh với hai phương pháp trước qua cùng một bài toán quản lý sinh viên. Với OOP, ta định nghĩa một **lớp** (class) mô tả khuôn mẫu cho sinh viên:

```
class SinhVien {
    // Dữ liệu (thuộc tính)
    String hoTen;
    String maSo;
    double diemToan;
    double diemVan;

    // Hành vi (phương thức)
    double tinhDiemTrungBinh() {
        return (diemToan + diemVan) / 2.0;
    }

    String xepLoai() {
        double dtb = tinhDiemTrungBinh();
        if (dtb >= 8.0) return "Giỏi";
        else if (dtb >= 6.5) return "Khá";
        else if (dtb >= 5.0) return "Trung bình";
        else return "Yếú";
    }
}
```

Từ khuôn mẫu này, mỗi sinh viên cụ thể (Nguyễn Văn A, Trần Thị B) là một **đối tượng** được tạo ra từ lớp `SinhVien`. Mỗi đối tượng mang dữ liệu riêng (tên, điểm số của chính người đó) nhưng chia sẻ cùng cách tính điểm và xếp loại được định nghĩa trong lớp.

```
SinhVien sv1 = new SinhVien();
sv1.hoTen = "Nguyêñ Văn A";
sv1.diemToan = 8.5;
```

```
sv1.diemVan = 7.0;
System.out.println(sv1.xepLoai());  // In ra: Khá
```

Cách tổ chức này phản ánh cách con người tư duy tự nhiên về thế giới: ta không nghĩ về một *danh sách tên* và một *danh sách điểm* riêng biệt, rồi dùng một hàm bên ngoài để nối chúng lại. Ta nghĩ về *từng sinh viên* như một thực thể trọn vẹn, có tên và có điểm, và biết cách tự đánh giá học lực của mình.

Bảng 1.1 tóm tắt sự khác biệt giữa ba phương pháp tiếp cận.

Bảng 1.1: So sánh ba phương pháp tiếp cận trong lập trình

Đặc điểm	**Lập trình tuyến tính**	**Lập trình hướng chức năng**	**Lập trình hướng đối tượng**
Đơn vị tổ chức	Dãy lệnh tuần tự	Hàm (function)	Đối tượng (object)
Dữ liệu và hành vi	Trộn lẫn trong dòng lệnh	Tách rời	Đóng gói trong đối tượng
Tái sử dụng	Sao chép mã	Gọi hàm	Kế thừa và giao diện
Quy mô phù hợp	Chương trình nhỏ (vài chục dòng)	Chương trình vừa (vài trăm dòng)	Hệ thống lớn (hàng nghìn dòng trở lên)
Tư duy thiết kế	Làm từng bước một	Chia thành các nhiệm vụ	Mô hình hóa thực thể trong thế giới thực

Lập trình hướng đối tượng không phải là sự thay thế tuyệt đối cho hai phương pháp trước. Trong thực tế, thân của mỗi phương thức trong OOP vẫn chứa các dãy lệnh tuần tự và các lời gọi hàm. OOP bổ sung thêm một **lớp tổ chức mới** bên trên — lớp đối tượng — giúp quản lý sự phức tạp của hệ thống lớn một cách tự nhiên và bền vững hơn.

Điều cần nhớ: Ba phương pháp tiếp cận không loại trừ lẫn nhau. OOP là *cách tổ chức ở cấp cao nhất*, giúp kiểm soát sự phức tạp của phần mềm lớn.

1.2 Các khái niệm cơ bản của lập trình hướng đối tượng

Để vận dụng được OOP, trước hết ta cần nắm vững sáu khái niệm nền tảng — **bộ từ vựng** của OOP: đối tượng, lớp, trừu tượng hoá, bao bọc, kế thừa và đa hình. Hãy bắt đầu từ khái niệm trung tâm nhất: **đối tượng**.

1.2.1 Đối tượng

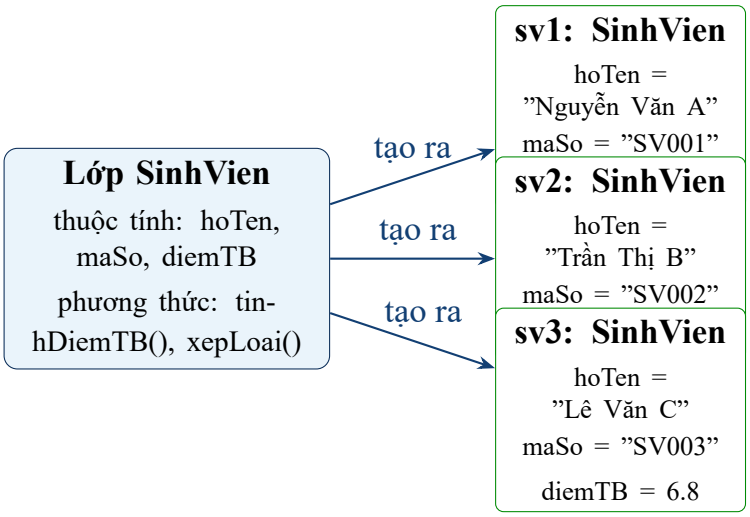
Mọi thực thể trong thế giới thực đều là các **đối tượng** (object) với hai đặc điểm cơ bản:

- **Trạng thái** (state): những gì đối tượng *biết* về chính nó. Ví dụ: sinh viên biết họ tên, mã số, điểm trung bình.
- **Hành vi** (behavior): những gì đối tượng *có thể làm*. Ví dụ: sinh viên có thể đăng ký môn học, tính điểm trung bình, xếp loại.

Trong OOP, trạng thái được lưu trong **thuộc tính** (biến), hành vi được cài đặt bằng **phương thức** (hàm). Ví dụ, đối tượng `sinhVienA` có trạng thái `hoTen = "Nguyễn Văn A"`, `diemTB = 7.5` và hành vi `tinhDiemTB()`, `xepLoai()`.

1.2.2 Lớp đối tượng

Nếu đối tượng là **thực thể cụ thể**, thì **lớp** (class) là **khuôn mẫu** (blueprint) dùng để tạo ra các đối tượng cùng loại. Một lớp định nghĩa các **thuộc tính** và **phương thức** chung cho mọi đối tượng thuộc lớp đó. Từ lớp `SinhVien`, ta có thể tạo ra nhiều đối tượng sinh viên, mỗi đối tượng có giá trị thuộc tính riêng nhưng dùng chung tập phương thức.



Hình 1.1: Quan hệ giữa lớp và đối tượng: một lớp là khuôn mẫu tạo ra nhiều đối tượng

Khi định nghĩa một lớp, ta cần chọn lọc những đặc điểm nào thực sự cần thiết cho bài toán. Quá trình này được gọi là **trừu tượng hoá** (abstraction) — giữ lại đặc điểm cốt yếu, loại bỏ chi tiết không liên quan. Ví dụ, trong hệ thống đào tạo, lớp `SinhVien` cần `diemTB` nhưng không cần `chieuCao`. Cùng một thực thể "sinh viên", ta sẽ trừu tượng hoá khác nhau tùy bài toán: hệ thống ký túc xá cần phòng ở, hệ thống y tế cần tiền sử bệnh.

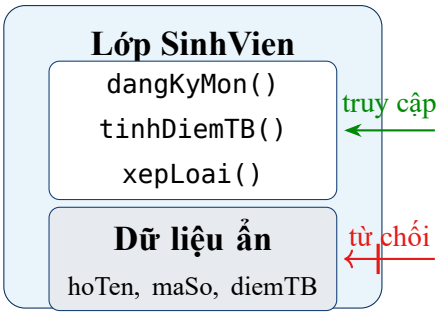
1.2.3 Bao bọc và che giấu thông tin

Bao bọc (encapsulation) là nguyên lý gói gọn dữ liệu và phương thức thao tác vào trong cùng một lớp, đồng thời **che giấu thông tin** (information hiding) — ẩn trạng thái bên trong, chỉ cho phép truy cập qua các phương thức công khai. Giống như tay ga xe máy: bạn vặn ga để tăng tốc mà không cần biết cơ chế bên trong động cơ.

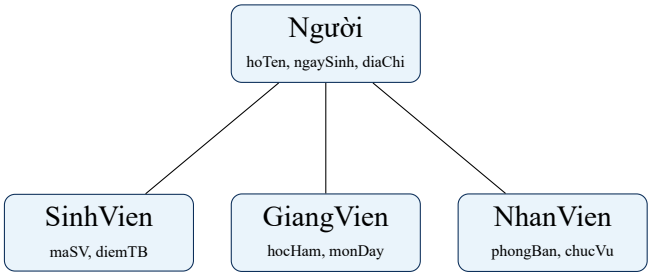
Bao bọc giúp bảo vệ dữ liệu khỏi gán giá trị không hợp lệ, dễ thay đổi cài đặt bên trong mà không ảnh hưởng mã bên ngoài, và giảm độ phức tạp khi sử dụng lớp.

1.2.4 Kế thừa

Kế thừa (inheritance) cho phép lớp mới thừa hưởng thuộc tính và phương thức từ lớp đã có, rồi bổ sung hoặc tuỳ chỉnh. Ví dụ: `SinhVien`, `GiangVien` đều kế thừa từ `Nguoi` (họ tên, ngày sinh), mỗi lớp con bổ sung đặc điểm riêng (mã SV, học hàm).



Hình 1.2: Nguyên lý bao bọc: dữ liệu được che giấu, chỉ truy cập qua phương thức công khai



Hình 1.3: Cây kế thừa: lớp con thừa hưởng thuộc tính từ lớp cha và bổ sung đặc điểm riêng

1.2.5 Đa hình

Đa hình (polymorphism) là khả năng cùng một lời gọi phương thức cho ra hành vi khác nhau tuỳ đối tượng thực thi. Ví dụ: `nguoi.ngheNhatBieu()` sẽ thực thi khác nhau khi `nguoi` là `SinhVien`, `GiangVien` hay `NhanVien`. Đa hình cho phép viết mã tổng quát ở mức lớp cha, không cần biết chi tiết lớp con.

Đa hình trong thực tế: Ổ cắm điện — mọi thiết bị (quạt, tivi, tủ lạnh) dùng chung một giao diện 220V, nhưng mỗi thiết bị phản ứng theo cách riêng. Đa hình trong OOP hoạt động theo cùng nguyên lý đó.

Bảng 1.2 tổng hợp sáu khái niệm và phép so sánh thực tế tương ứng.

Điều cần nhớ: Sáu khái niệm trên không phải là sáu "tính năng" rời rạc của OOP, mà là sáu *cách nhìn* về cùng một hệ thống. Một thiết kế hướng đối tượng tốt là thiết kế vận dụng hài hoà cả sáu khái niệm: đối tượng được tạo từ lớp, lớp được thiết kế qua trừu tượng hoá, dữ liệu được bảo vệ bằng bao bọc, mã nguồn được tái sử dụng qua kế thừa, và hệ thống trở nên linh hoạt nhờ đa hình.

1.3 Cách tiếp cận và giải quyết bài toán theo phương pháp hướng đối tượng

Phần này trả lời câu hỏi: làm thế nào để **vận dụng** sáu khái niệm OOP vào giải quyết bài toán thực tế? Quy trình phân tích và thiết kế hướng đối tượng gồm sáu bước, tạo thành **khung làm việc** (framework) áp dụng xuyên suốt giáo trình. Ta sẽ minh họa qua bài toán: **xây dựng hệ thống quản lý thư viện**.

Bảng 1.2: Sáu khái niệm cơ bản của OOP và phép so sánh thực tế

Khái niệm	Ý nghĩa trong OOP	So sánh thực tế
Đối tượng	Thực thể mang trạng thái và hành vi	Một chiếc xe máy cụ thể với biển số, màu sơn, mức xăng
Lớp	Khuôn mẫu định nghĩa thuộc tính và phương thức cho một nhóm đối tượng	Bản vẽ thiết kế xe máy — từ đó sản xuất ra hàng nghìn chiếc
Trừu tượng hoá	Giữ lại đặc điểm cốt yếu, loại bỏ chi tiết không liên quan	Mô tả sinh viên cho hệ thống đào tạo: cần điểm số, không cần chiều cao
Bao bọc và che giấu	Gói dữ liệu và phương thức trong lớp; ẩn chi tiết bên trong	Tay ga xe máy: vặn là tăng tốc, không cần biết cơ chế bên trong
Kế thừa	Lớp con thừa hưởng thuộc tính và phương thức từ lớp cha	Sinh viên, giảng viên đều *là* người, có chung họ tên, ngày sinh
Đa hình	Cùng một lời gọi, hành vi khác nhau tuỳ đối tượng	Cùng một ổ điện, quạt thì quay, tivi thì phát hình, tủ lạnh thì làm mát

1.3.1 Mô tả bài toán

Bước đầu tiên — cũng là bước quyết định chất lượng của toàn bộ thiết kế — là **hiểu đúng bài toán**. Nghe có vẻ hiển nhiên, nhưng trong thực tế, đây là bước thường bị xem nhẹ nhất. Một thiết kế xuất sắc cho *sai bài toán* là một thiết kế vô giá trị.

Nguyên tắc: Hãy dành thời gian mô tả bài toán bằng ngôn ngữ tự nhiên, rõ ràng và đầy đủ. Việc viết ra buộc ta phải suy nghĩ thấu đáo. Nếu không thể mô tả bài toán trong vài câu rành mạch, rất có thể ta chưa thực sự hiểu nó.

Với hệ thống quản lý thư viện, bản mô tả bài toán có thể như sau:

> Thư viện trường đại học cần một hệ thống để quản lý việc mượn và trả sách. Thư viện có nhiều đầu sách, mỗi đầu sách có thể có nhiều bản sao. Sinh viên và giảng viên đều có thể mượn sách. Mỗi lần mượn, hệ thống cần ghi nhận ai mượn, mượn sách gì, vào ngày nào và hạn trả là khi nào. Khi sách được trả, hệ thống cần cập nhật trạng thái của bản sao đó. Hệ thống cũng cần có khả năng tra cứu sách theo tên, tác giả hoặc mã số, và thông báo khi có sách quá hạn chưa trả.

Bản mô tả này còn nhiều điểm chưa rõ (ví dụ: sinh viên được mượn tối đa bao nhiêu sách? Thời hạn mượn là bao lâu? Có phân biệt thời hạn giữa sinh viên và giảng viên không?). Những câu hỏi này sẽ được làm rõ ở bước tiếp theo.

1.3.2 Đặc tả yêu cầu

Từ bản mô tả bài toán, bước thứ hai là **đặc tả yêu cầu** (requirements specification) — liệt kê một cách có hệ thống những gì hệ thống *phải làm được*. Yêu cầu thường được chia thành hai loại:

- **Yêu cầu chức năng** (functional requirements): những việc hệ thống phải thực hiện — các tính năng, các thao tác mà người dùng có thể làm.
- **Yêu cầu phi chức năng** (non-functional requirements): những ràng buộc về chất lượng, hiệu năng, bảo mật, hoặc môi trường vận hành.

Với hệ thống quản lý thư viện, ta có thể đặc tả:

Yêu cầu chức năng:

1. Quản lý đầu sách: thêm mới, cập nhật thông tin, tìm kiếm theo tên/tác giả/mã số.
2. Quản lý bản sao: theo dõi từng bản sao của mỗi đầu sách và trạng thái (có sẵn/đang mượn/hỏng).
3. Quản lý người mượn: lưu thông tin sinh viên và giảng viên; mỗi người mượn có hạn mức và thời hạn mượn riêng.
4. Mượn sách: ghi nhận giao dịch mượn, kiểm tra điều kiện (người mượn còn hạn mức không, sách có sẵn không), ấn định ngày hẹn trả.
5. Trả sách: ghi nhận giao dịch trả, cập nhật trạng thái bản sao, tính tiền phạt nếu quá hạn.
6. Cảnh báo quá hạn: liệt kê các giao dịch đã quá hạn mà chưa trả.

Yêu cầu phi chức năng (ví dụ):

- Hệ thống phải phản hồi mọi thao tác tra cứu trong dưới 2 giây.
- Dữ liệu phải được lưu trữ bền vững, không mất khi mất điện.
- Giao diện đơn giản, dùng được cho nhân viên thư viện không có kỹ năng công nghệ thông tin chuyên sâu.

Đặc tả yêu cầu giúp ta biết *cần làm gì*. Bước tiếp theo sẽ trả lời câu hỏi *có những thực thể nào* trong bài toán — đây chính là lúc kỹ năng trừu tượng hoá phát huy vai trò.

1.3.3 Trích chọn đối tượng

Đây là bước chuyển từ "bài toán" sang "thiết kế hướng đối tượng". Nhiệm vụ của bước này là **đọc kỹ bản mô tả và đặc tả yêu cầu**, từ đó xác định các **đối tượng** (object) và **lớp** (class) tiềm năng.

Một kỹ thuật đơn giản và hiệu quả là **gạch chân danh từ** (noun identification): đọc lướt qua bản mô tả, đánh dấu mọi danh từ hoặc cụm danh từ. Mỗi danh từ là một ứng viên cho lớp hoặc thuộc tính.

Áp dụng vào bản mô tả thư viện, ta thu được các danh từ sau:

- Thư viện, trường đại học, hệ thống
- **Đầu sách**, **bản sao** (của sách)
- **Sinh viên**, **giảng viên**, người mượn
- **Giao dịch** (mượn/trả), ngày mượn, hạn trả, ngày trả
- Tên sách, tác giả, mã số

Không phải mọi danh từ đều trở thành lớp. Ta cần lọc lại dựa trên các tiêu chí sau:

1. **Danh từ nào thực sự cần lưu trữ thông tin?** Nếu một danh từ chỉ cần một giá trị đơn giản (ví dụ: tên sách, tác giả), nó nên là *thuộc tính* của một lớp, chứ không phải một lớp riêng.

2. **Danh từ nào là một thực thể độc lập với vòng đời riêng?** Một đầu sách được thêm vào thư viện, tồn tại qua nhiều lần mượn/trả — đó là một thực thể. Một giao dịch mượn sách được tạo ra, rồi kết thúc khi sách được trả — đó cũng là một thực thể.
3. **Danh từ nào có hành vi riêng?** Sinh viên có hành vi "mượn sách" và "trả sách"; đầu sách có hành vi "kiểm tra còn bản sao nào không".
4. **Có danh từ nào thực chất là cùng một khái niệm?** "Sinh viên" và "giảng viên" có thể gộp thành một khái niệm khái quát hơn: "Người mượn".

Sau khi lọc, ta xác định được các **lớp chính** cho hệ thống thư viện:

Bảng 1.3: Kết quả trích chọn đối tượng cho hệ thống quản lý thư viện

Lớp	Lý do là lớp	Trở thành thuộc tính
DauSach	Cần quản lý thông tin: tên, tác giả, mã số, danh sách bản sao	Tên tác giả, mã số
BanSao	Mỗi bản sao có trạng thái riêng (có sẵn/đang mượn)	Mã vạch, trạng thái
NguoiMuon	Sinh viên và giảng viên: chung đặc điểm, khác hạn mức và thời hạn	Họ tên, mã số, loại
GiaoDich	Mỗi lần mượn/trả cần được ghi nhận riêng	Ngày mượn, hạn trả, ngày trả

Sau bước này, ta đã có một danh sách sơ bộ các lớp. Bước tiếp theo sẽ làm rõ từng lớp: mỗi lớp mang những thuộc tính gì và có thể thực hiện những hành vi gì.

1.3.4 Mô hình hoá lớp đối tượng

Với danh sách lớp đã có, bước này **chi tiết hoá** từng lớp: xác định thuộc tính và phương thức. Đây chính là lúc các khái niệm *trừu tượng hoá* (giữ gì, bỏ gì) và *bao bọc* (ẩn gì, lộ gì) được thực hành một cách cụ thể.

Với mỗi lớp, hãy tự hỏi ba câu hỏi:

1. **Lớp này cần biết gì?** → thuộc tính (attributes).
2. **Lớp này cần làm được gì?** → phương thức (methods).
3. **Người dùng lớp này cần thấy gì?** → giao diện công khai (public interface).

Áp dụng cho lớp `DauSach`:

Lớp DauSach
Thuộc tính: `maSo`, `tenSach`, `tacGia`, `namXuatBan`, `danhSachBanSao`
Phương thức: `themBanSao()`, `xoaBanSao()`, `soBanSaoCoSan()` — trả về số bản sao đang ở trạng thái "có sẵn"
Che giấu: `danhSachBanSao` được ẩn; bên ngoài không được phép trực tiếp thêm/xoá bản sao khỏi danh sách mà phải qua phương thức `themBanSao()` / `xoaBanSao()` để đảm bảo tính toàn vẹn dữ liệu.

Tương tự cho lớp `NguoiMuon`:

Lớp NguoiMuon
Thuộc tính: `maSo`, `hoTen`, `loai` (SinhVien / GiangVien), `danhSachDangMuon`
Phương thức: `muonSach()`, `traSach()`, `soSachDangMuon()`
Che giấu: `danhSachDangMuon` được ẩn; số sách đang mượn được kiểm tra qua phương thức, không cho phép gán trực tiếp.

Lưu ý rằng ở bước này, ta chưa cần viết code. Các mô tả trên có thể được ghi lại dưới dạng **thẻ CRC** (Class-Responsibility-Collaboration) — một công cụ đơn giản dùng thẻ giấy để mô tả tên lớp, trách nhiệm và các lớp cộng tác. Đây là kỹ thuật phổ biến trong các buổi thảo luận thiết kế nhóm.

1.3.5 Thiết kế tổng quát

Sau khi đã có các lớp với thuộc tính và phương thức, bước tiếp theo là xác định **mối quan hệ** giữa chúng. Các lớp không tồn tại biệt lập — chúng tương tác và liên kết với nhau để tạo thành một hệ thống hoàn chỉnh. Bước này tập trung vào **kiến trúc tổng thể** của hệ thống.

Có ba loại quan hệ chính cần xác định:

1. **Kết hợp** (association): hai lớp có liên quan với nhau một cách lỏng lẻo. Ví dụ: `NguoiMuon` *mượn* `BanSao` — đây là quan hệ kết hợp, được hiện thực qua lớp `GiaoDich`.
2. **Tập hợp** (aggregation): một đối tượng *chứa* đối tượng khác, nhưng đối tượng bị chứa có thể tồn tại độc lập. Ví dụ: `DauSach` chứa nhiều `BanSao` — nhưng một `BanSao` có thể được chuyển sang `DauSach` khác (khi nhập nhầm).
3. **Kế thừa** (inheritance): quan hệ *is-a* giữa lớp cha và lớp con. Ví dụ: `SinhVien` và `GiangVien` đều *là* `NguoiMuon` — chúng kế thừa các thuộc tính chung như `hoTen`, `maSo` nhưng khác nhau ở hạn mức và thời hạn mượn.

Với hệ thống thư viện, sơ đồ quan hệ giữa các lớp được thể hiện trong Hình 1.4.

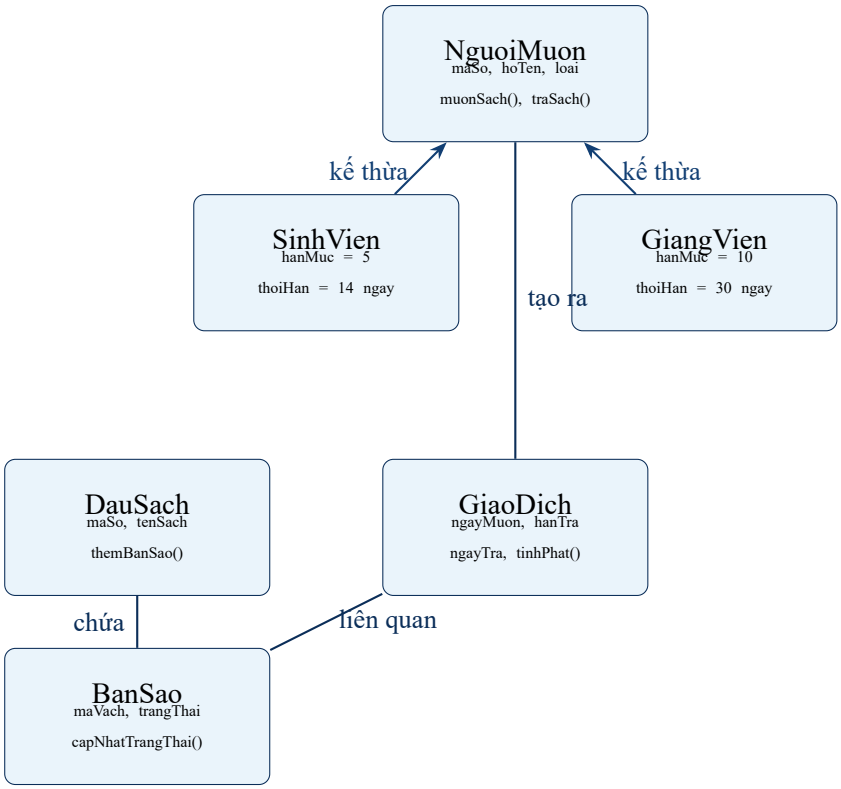
Bước thiết kế tổng quát cho ta **bức tranh toàn cảnh** — ai liên quan đến ai, ai kế thừa ai, ai chứa ai. Từ bức tranh này, bước cuối cùng sẽ đi vào chi tiết từng tương tác.

1.3.6 Thiết kế chi tiết

Bước cuối cùng là **thiết kế chi tiết** (detailed design): mô tả chính xác cách các đối tượng tương tác với nhau để thực hiện một chức năng cụ thể. Ở bước này, ta dùng **biểu đồ tuần tự** (sequence diagram) — một công cụ trực quan thể hiện luồng thông điệp giữa các đối tượng theo thời gian.

Hãy xét chức năng "mượn sách" — chức năng phức tạp nhất trong hệ thống thư viện. Khi một sinh viên đến quầy và yêu cầu mượn một bản sao của đầu sách "Lập trình Java", chuỗi sự kiện diễn ra như sau:

1. **Kiểm tra người mượn:** Hệ thống kiểm tra `sinhVien.soSachDangMuon()` có vượt hạn mức không.
2. **Kiểm tra bản sao:** Hệ thống kiểm tra bản sao được yêu cầu có đang ở trạng thái "có sẵn" không.
3. **Tạo giao dịch:** Nếu cả hai điều kiện đều thoả mãn, một đối tượng `GiaoDich` mới được tạo ra với ngày mượn là hôm nay và hạn trả được tính tự động dựa trên loại



Hình 1.4: Sơ đồ thiết kế tổng quát của hệ thống quản lý thư viện

người mượn.

4. **Cập nhật trạng thái:** Bản sao được chuyển sang trạng thái "đang mượn"; giao dịch được thêm vào danh sách đang mượn của sinh viên.
5. **Trả về kết quả:** Hệ thống thông báo mượn thành công kèm ngày hẹn trả.

Hình 1.5 minh họa luồng tương tác này dưới dạng biểu đồ tuần tự.

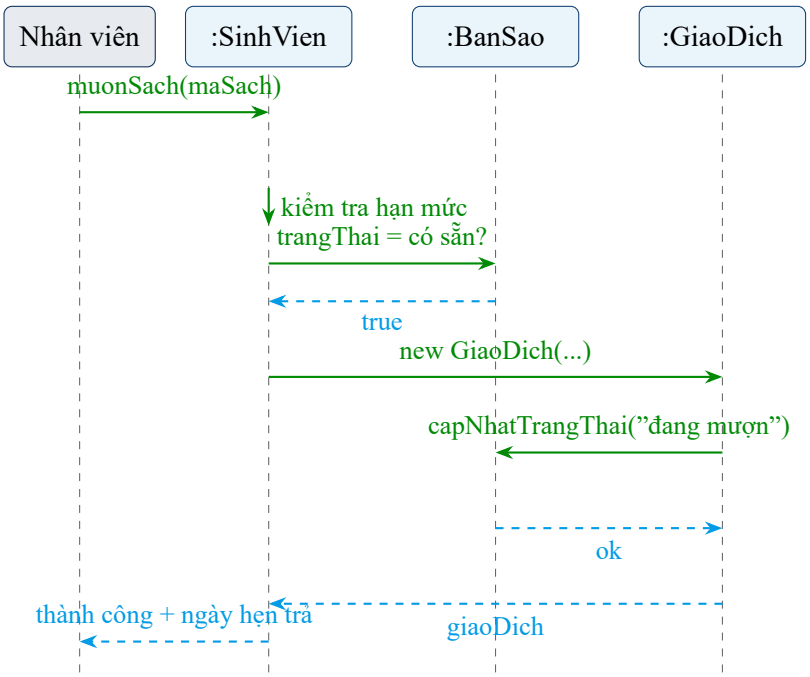
Kết thúc bước thiết kế chi tiết, ta đã có đủ thông tin để bắt đầu viết code. Lưu ý rằng trong toàn bộ quy trình sáu bước trên, ta chưa hề viết một dòng Java nào — và đó chính là điểm mạnh của phương pháp hướng đối tượng: **thiết kế trước, cài đặt sau**. Một thiết kế tốt giúp việc cài đặt trở nên rõ ràng và ít lỗi hơn rất nhiều.

Điều cần nhớ: Sáu bước trên không phải là một quy trình cứng nhắc. Trong thực tế, ta thường xuyên quay lui: khi thiết kế chi tiết phát hiện một lớp còn thiếu, ta quay lại bước trích chọn đối tượng; khi đặc tả yêu cầu thấy mâu thuẫn với mô tả bài toán, ta quay lại bước đầu để làm rõ. Thiết kế hướng đối tượng là một quá trình **lặp** (iterative) và **tăng dần** (incremental).

Bảng 1.4 tóm tắt toàn bộ quy trình — đây cũng chính là **khung làm việc** mà ta sẽ áp dụng trong mọi case study từ các chương sau.

1.4 Một số ngôn ngữ lập trình hướng đối tượng

Lập trình hướng đối tượng là một **phương pháp tư duy**, không phải cú pháp cụ thể. Phần này điểm qua bốn ngôn ngữ OOP phổ biến và lý do Java được chọn cho giáo trình này.



Hình 1.5: Biểu đồ tuần tự cho chức năng “mượn sách” — mô tả chi tiết luồng tương tác

Bảng 1.4: Tổng hợp quy trình sáu bước trong phân tích và thiết kế hướng đối tượng

Bước	Tên bước	Câu hỏi chính	Kết quả đầu ra
1	Mô tả bài toán	Bài toán là gì?	Văn bản mô tả bằng ngôn ngữ tự nhiên
2	Đặc tả yêu cầu	Hệ thống phải làm được gì?	Danh sách yêu cầu chức năng và phi chức năng
3	Trích chọn đối tượng	Có những thực thể nào?	Danh sách lớp ứng viên
4	Mô hình hoá lớp	Mỗi lớp biết gì, làm được gì?	Thuộc tính và phương thức của từng lớp
5	Thiết kế tổng quát	Các lớp liên kết với nhau ra sao?	Sơ đồ lớp với quan hệ kết hợp, tập hợp, kế thừa
6	Thiết kế chi tiết	Các đối tượng tương tác thế nào cho từng chức năng?	Biểu đồ tuần tự cho các chức năng chính

C++ ra đời năm 1979 do Bjarne Stroustrup phát triển, mở rộng ngôn ngữ C bằng các tính năng hướng đối tượng. C++ nổi bật với **hiệu năng cao** và **kiểm soát chặt chẽ tài nguyên** qua con trỏ, thường dùng trong game, phần mềm nhúng, và hệ điều hành. Tuy nhiên, C++ khá phức tạp đối với người mới học.

Java ra đời năm 1995 do James Gosling và Sun Microsystems phát triển, với triết lý "viết một lần, chạy mọi nơi" (WORA). Java biên dịch ra **mã byte** chạy trên **máy ảo Java** (JVM), giúp chương trình chạy được trên mọi hệ điều hành. Java loại bỏ con trỏ, dùng cơ chế **thu gom rác** (garbage collection) tự động, và được thiết kế **thuần hướng đối tượng**. Java được dùng rộng rãi trong ứng dụng doanh nghiệp, web, Android, và là ngôn ngữ được dạy nhiều nhất trong các trường đại học.

C# được Microsoft giới thiệu năm 2000, kết hợp sức mạnh của C++ và sự thanh lịch của Java. C# bổ sung các tính năng hiện đại như **thuộc tính** (properties) và **LINQ**. C# là ngôn ngữ chính cho nền tảng .NET, được dùng trong ứng dụng Windows, doanh nghiệp, và phát triển game với Unity.

Python được Guido van Rossum tạo ra năm 1991, là ngôn ngữ **đa phương pháp** (multi-paradigm) với cú pháp ngắn gọn, dễ học. Python mạnh trong khoa học dữ liệu, AI, học máy và tự động hóa. Tuy nhiên, Python không có cơ chế kiểm soát truy cập chặt chẽ như Java, nên không phải lựa chọn tốt nhất để dạy OOP cho người mới bắt đầu.

Bảng 1.5 tổng hợp và so sánh bốn ngôn ngữ theo các tiêu chí chính.

Bảng 1.5: So sánh bốn ngôn ngữ lập trình hướng đối tượng phổ biến

Tiêu chí	C++	Java	C#	Python
Năm ra đời	1979	1995	2000	1991
Biên dịch	Ra mã máy trực tiếp	Ra bytecode, chạy trên JVM	Ra CIL, chạy trên .NET CLR	Thông dịch (có thể biên dịch JIT)
Quản lý bộ nhớ	Thủ công (có con trỏ)	Tự động (garbage collection)	Tự động (garbage collection)	Tự động (garbage collection)
Đa kế thừa	Có (lớp)	Không (dùng interface)	Không (dùng interface)	Có (lớp)
Thuần OOP	Không (hỗ trợ lập trình thủ tục)	Có (mọi thứ trong lớp)	Có	Không (đa phương pháp)
Độ khó học	Cao	Trung bình	Trung bình	Thấp
Ứng dụng chính	Game, hệ thống nhúng, hệ điều hành	Doanh nghiệp, web, Android, giáo dục	Windows, game (Unity), doanh nghiệp	Khoa học dữ liệu, AI, tự động hóa, web

Tại sao giáo trình này chọn Java? Bốn lý do chính:

1. **Thuần hướng đối tượng:** Java yêu cầu mọi thứ nằm trong lớp, buộc người học tư duy OOP ngay từ đầu. Không có con trỏ và quản lý bộ nhớ thủ công, giảm tải nhận thức cho người mới học.

2. **Cú pháp chặt chẽ:** Khai báo kiểu tường minh, kiểm soát truy cập nghiêm ngặt — giúp hình thành thói quen lập trình chuyên nghiệp.
3. **Hệ sinh thái phong phú:** Hàng nghìn thư viện, IDE miễn phí (Eclipse, IntelliJ IDEA), cộng đồng toàn cầu lớn.
4. **Nhu cầu thị trường cao:** Java liên tục nằm trong nhóm ngôn ngữ được tuyển dụng nhiều nhất.

Điều cần nhớ: Không có ngôn ngữ nào là "tốt nhất" một cách tuyệt đối. Mỗi ngôn ngữ là một công cụ phù hợp với những bài toán và ngữ cảnh khác nhau. Sau khi đã nắm vững các nguyên lý OOP qua Java, việc chuyển sang học một ngôn ngữ OOP khác như C++ hay C# sẽ trở nên dễ dàng hơn rất nhiều, vì các khái niệm cốt lõi là giống nhau — chỉ khác về cú pháp và một số quy ước riêng.

1.5 Giới thiệu một số Editor biên soạn chương trình Java

Câu hỏi đầu tiên của người mới bắt đầu: **viết code Java ở đâu?** Một **môi trường phát triển tích hợp** (IDE — Integrated Development Environment) cung cấp bộ công cụ toàn diện: soạn thảo có gợi ý thông minh, biên dịch/chạy một nút bấm, và gỡ lỗi trực quan. Phần này giới thiệu các IDE phổ biến cho lập trình Java.

1.5.1 Eclipse

Eclipse là một trong những IDE mã nguồn mở lâu đời và phổ biến nhất cho lập trình Java. Được phát triển bởi Eclipse Foundation, Eclipse ra đời từ năm 2001 và nhanh chóng trở thành lựa chọn hàng đầu trong cả môi trường giáo dục lẫn doanh nghiệp.

Những điểm mạnh chính của Eclipse:

- **Miễn phí và mã nguồn mở:** Eclipse hoàn toàn miễn phí, có thể tải về từ `https://www.eclipse.org/downloads/`.
- **Hệ sinh thái plugin khổng lồ:** Eclipse có thể mở rộng chức năng thông qua hàng nghìn plugin — từ hỗ trợ các ngôn ngữ khác (C++, Python, PHP) đến công cụ quản lý cơ sở dữ liệu, thiết kế giao diện, và tích hợp với các hệ thống kiểm soát phiên bản như Git.
- **Công cụ tái cấu trúc mạnh mẽ:** Eclipse cung cấp các tính năng refactoring tự động như đổi tên biến/hàm/lớp trên toàn dự án, trích xuất phương thức, di chuyển lớp giữa các gói — giúp duy trì mã nguồn sạch và nhất quán.
- **Hỗ trợ gỡ lỗi trực quan:** Debugger của Eclipse cho phép đặt điểm dừng (breakpoint), chạy từng bước, và kiểm tra giá trị biến trong thời gian thực — công cụ không thể thiếu khi truy vết lỗi.

Eclipse có nhược điểm là giao diện hơi phức tạp đối với người mới bắt đầu và thời gian khởi động khá lâu trên các máy cấu hình thấp. Dù vậy, đây vẫn là một lựa chọn rất tốt, đặc biệt trong môi trường đại học — nơi tính miễn phí và khả năng mở rộng được ưu tiên hàng đầu.

1.5.2 NetBeans

NetBeans là một IDE mã nguồn mở khác, ban đầu được Sun Microsystems phát triển (sau này được Oracle tiếp quản cùng với Java) và hiện nay do Apache Software

Foundation duy trì. NetBeans được thiết kế với triết lý **"mọi thứ có sẵn ngay từ đầu"** (out-of-the-box) — khác với Eclipse vốn cần cài thêm plugin cho nhiều tính năng.

Những điểm mạnh chính của NetBeans:

- **Cấu hình đơn giản:** NetBeans đi kèm sẵn hầu hết các công cụ cần thiết cho phát triển Java mà không cần cài đặt thêm. Sau khi tải về từ `https://netbeans.apache.org/`, bạn có thể bắt đầu viết code ngay lập tức.
- **Trình thiết kế giao diện kéo-thả:** NetBeans có công cụ thiết kế GUI (Graphical User Interface) trực quan, cho phép tạo giao diện người dùng bằng cách kéo và thả các thành phần như nút bấm, ô văn bản, bảng biểu — rất hữu ích khi học về lập trình giao diện với Java Swing hoặc JavaFX.
- **Tích hợp chặt chẽ với Java:** NetBeans từng là IDE chính thức được Oracle khuyến nghị cho Java, do đó việc tích hợp với các phiên bản JDK mới nhất luôn được đảm bảo nhanh chóng và ổn định.
- **Hỗ trợ nhiều ngôn ngữ:** Ngoài Java, NetBeans còn hỗ trợ tốt cho HTML, CSS, JavaScript, PHP, và C/C++, khiến nó trở thành công cụ đa năng cho sinh viên cần làm việc với nhiều công nghệ khác nhau trong quá trình học.

NetBeans có xu hướng tiêu tốn nhiều bộ nhớ hơn Eclipse và cộng đồng người dùng nhỏ hơn, nhưng bù lại, trải nghiệm ban đầu mượt mà hơn — một lợi thế đáng kể cho người mới bắt đầu.

1.5.3 Một số IDE thông dụng khác

Ngoài Eclipse và NetBeans, thị trường IDE cho Java còn có nhiều lựa chọn khác. Dưới đây là ba công cụ nổi bật mà bạn nên biết.

IntelliJ IDEA do JetBrains phát triển, hiện là IDE được cộng đồng Java chuyên nghiệp đánh giá cao nhất. IntelliJ nổi tiếng với khả năng **gợi ý mã thông minh** (intelligent code completion) — IDE có thể dự đoán ý định của người lập trình và đề xuất chính xác đoạn mã cần viết tiếp theo. IntelliJ có hai phiên bản: **Community Edition** (miễn phí, đủ dùng cho hầu hết nhu cầu học tập và phát triển cơ bản) và **Ultimate Edition** (trả phí, bổ sung hỗ trợ cho phát triển web doanh nghiệp, cơ sở dữ liệu, và nhiều framework nâng cao). Đối với sinh viên, JetBrains cung cấp giấy phép Ultimate Edition miễn phí thông qua chương trình **JetBrains Student License** — bạn chỉ cần đăng ký bằng email trường đại học. Tải về tại `https://www.jetbrains.com/idea/download/`.

Visual Studio Code (VS Code) của Microsoft là một trình soạn thảo mã nguồn nhẹ (code editor), không phải là một IDE đầy đủ. Tuy nhiên, nhờ hệ sinh thái extension phong phú, VS Code có thể được biến thành một môi trường phát triển Java gần như hoàn chỉnh. Ưu điểm lớn nhất của VS Code là **nhẹ, nhanh** và **đa năng** — cùng một công cụ có thể dùng cho Java, Python, JavaScript, viết tài liệu Markdown, LaTeX, và nhiều tác vụ khác. Đối với sinh viên cần làm việc với nhiều ngôn ngữ trong các môn học khác nhau, VS Code là một lựa chọn rất thực tế. Tải về tại `https://code.visualstudio.com/`.

BlueJ là một IDE được thiết kế **đặc biệt cho mục đích giảng dạy** lập trình hướng đối tượng. Được phát triển bởi Đại học Kent (Anh) và Đại học Deakin (Úc), BlueJ có giao diện cực kỳ đơn giản và độc đáo: nó hiển thị các lớp dưới dạng sơ đồ UML trực quan, cho phép sinh viên tạo đối tượng và gọi phương thức tương tác ngay trên giao diện đồ họa mà không cần viết mã `main`. BlueJ rất phù hợp để **thực hành các khái niệm**

OOP cơ bản trong vài tuần đầu làm quen, nhưng không được thiết kế cho phát triển dự án thực tế.

Bảng 1.6 tổng hợp và so sánh các IDE theo các tiêu chí phù hợp với sinh viên mới bắt đầu.

Bảng 1.6: So sánh các IDE cho lập trình Java

Tiêu chí	Eclipse	NetBeans	IntelliJ IDEA CE	VS Code
Loại	IDE đầy đủ	IDE đầy đủ	IDE đầy đủ	Code Editor
Giấy phép	Miễn phí, mã nguồn mở	Miễn phí, mã nguồn mở	Miễn phí (CE) / Trả phí (Ultimate)	Miễn phí, mã nguồn mở
Độ nặng	Trung bình–nặng	Trung bình–nặng	Nhẹ–trung bình	Nhẹ
Dễ bắt đầu	Trung bình	Dễ	Dễ	Cần cài extension
Tính năng nổi bật	Plugin phong phú, refactoring mạnh	Thiết kế GUI kéo-thả, cấu hình sẵn	Gợi ý mã thông minh nhất	Đa ngôn ngữ, nhẹ và nhanh
Phù hợp nhất cho	Học OOP, dự án vừa và lớn	Học OOP, làm quen GUI	Học và phát triển chuyên nghiệp	Làm việc đa ngôn ngữ, máy cấu hình thấp

Nên chọn IDE nào? Đối với giáo trình này, không có yêu cầu bắt buộc về IDE — bạn có thể chọn bất kỳ công cụ nào trong số trên. Lời khuyên cho người mới bắt đầu là:

- Nếu máy tính của bạn có cấu hình tốt và bạn muốn một trải nghiệm **đầy đủ, chuyên nghiệp, cài xong dùng ngay**: chọn **IntelliJ IDEA Community Edition**.
- Nếu bạn muốn một IDE **miễn phí hoàn toàn, mạnh mẽ, và có cộng đồng lớn** tại Việt Nam: chọn **Eclipse**.
- Nếu máy tính có cấu hình thấp hoặc bạn cần dùng chung một công cụ cho nhiều môn học khác nhau: chọn **VS Code** kèm Java Extension Pack.

Lưu ý: Tất cả code ví dụ trong giáo trình này được viết và kiểm tra bằng Java với JDK 17 trở lên. Bất kỳ IDE nào kể trên, khi được cấu hình đúng với JDK phù hợp, đều có thể biên dịch và chạy được toàn bộ ví dụ và bài tập trong sách.

Tự kiểm tra: Hãy tải và cài đặt **một** IDE mà bạn chọn. Sau khi cài đặt, tạo một dự án Java mới và thử gõ chương trình đơn giản sau, rồi chạy để xác nhận mọi thứ hoạt động bình thường:

```
public class XinChao {
    public static void main(String[] args) {
        System.out.println("Xin chào Java!");
    }
}
```

Nếu bạn thấy dòng chữ "Xin chào Java!" xuất hiện trên màn hình, bạn đã sẵn sàng cho chương tiếp theo — nơi chúng ta sẽ bắt đầu tìm hiểu Java một cách có hệ thống.

1.6 Tổng kết chương

Chương mở đầu này đã đặt nền móng tư duy cho toàn bộ giáo trình. Ta bắt đầu bằng việc hiểu lập trình hướng đối tượng *đến từ đâu* — đó là kết quả của một quá trình tiến hóa tự nhiên từ lập trình tuyến tính và lập trình hướng chức năng, được thúc đẩy bởi nhu cầu quản lý sự phức tạp ngày càng tăng của phần mềm. Mỗi phương pháp đều có chỗ đứng riêng, và OOP không thay thế chúng mà bổ sung một lớp tổ chức mới bên trên.

Từ nền tảng đó, ta đã làm quen với **sáu khái niệm trụ cột** — bộ từ vựng của OOP — gồm: đối tượng (thực thể mang trạng thái và hành vi), lớp (khuôn mẫu tạo đối tượng), trừu tượng hoá (giữ cốt yếu, bỏ chi tiết thừa), bao bọc và che giấu thông tin (bảo vệ dữ liệu, ẩn chi tiết cài đặt), kế thừa (lớp con thừa hưởng từ lớp cha), và đa hình (cùng lời gọi, hành vi khác nhau tùy đối tượng). Sáu khái niệm này không đứng riêng rẽ mà liên kết chặt chẽ với nhau.

Tiếp theo, ta đã học cách vận dụng các khái niệm đó thông qua **quy trình sáu bước** phân tích và thiết kế hướng đối tượng: từ mô tả bài toán, đặc tả yêu cầu, trích chọn đối tượng, mô hình hoá lớp, thiết kế tổng quát, cho đến thiết kế chi tiết. Quy trình này — được minh họa xuyên suốt qua ví dụ hệ thống quản lý thư viện — là khung làm việc mà ta sẽ áp dụng lặp đi lặp lại trong các chương sau.

Cuối cùng, ta đã điểm qua bốn ngôn ngữ OOP phổ biến (C++, Java, C#, Python), so sánh đặc điểm và lĩnh vực ứng dụng của từng ngôn ngữ, đồng thời hiểu được lý do Java được chọn làm ngôn ngữ minh họa chính. Phần giới thiệu các IDE đã trang bị cho bạn công cụ thực hành cần thiết để bắt đầu viết những dòng code Java đầu tiên.

Từ chương tiếp theo, ta sẽ đi sâu vào ngôn ngữ Java một cách có hệ thống: từ lịch sử và hệ sinh thái Java, cấu trúc chương trình, cho đến các thành phần cơ sở của ngôn ngữ. Những khái niệm OOP được giới thiệu trong chương này sẽ dần được hiện thực hoá bằng code Java cụ thể trong suốt phần còn lại của giáo trình.

Ôn tập cuối chương

Tóm tắt kiến thức cần nhớ

- **Ba phương pháp tiếp cận trong lập trình:** Lập trình tuyến tính tổ chức chương trình thành chuỗi lệnh tuần tự; lập trình hướng chức năng tổ chức thành các hàm độc lập, tái sử dụng được; lập trình hướng đối tượng đóng gói dữ liệu và hành vi thành các đối tượng. Ba phương pháp bổ trợ cho nhau, không loại trừ lẫn nhau.
- **Đối tượng và lớp:** Đối tượng là thực thể mang trạng thái (thuộc tính) và hành vi (phương thức). Lớp là khuôn mẫu định nghĩa cấu trúc chung cho một nhóm đối tượng — một lớp có thể tạo ra nhiều đối tượng.
- **Trừu tượng hoá:** Quá trình giữ lại những đặc điểm cốt yếu của thực thể và loại bỏ chi tiết không liên quan đến bài toán. Cùng một thực thể có thể được trừu tượng hoá khác nhau tùy theo bài toán.
- **Bao bọc và che giấu thông tin:** Gói dữ liệu và phương thức trong lớp; ẩn chi tiết cài đặt bên trong, chỉ cho phép truy cập qua giao diện công khai. Bảo vệ dữ liệu, dễ bảo trì, giảm độ phức tạp.

- **Kế thừa:** Lớp con thừa hưởng thuộc tính và phương thức từ lớp cha, thể hiện quan hệ *is-a*. Giúp tái sử dụng mã nguồn và phản ánh phân cấp tự nhiên của thế giới thực.
- **Đa hình:** Cùng một lời gọi phương thức có thể cho ra hành vi khác nhau tùy đối tượng thực thi. Cho phép viết mã tổng quát ở mức lớp cha nhưng vận hành linh hoạt theo từng lớp con.
- **Quy trình sáu bước thiết kế OOP:** Mô tả bài toán → Đặc tả yêu cầu → Trích chọn đối tượng → Mô hình hoá lớp → Thiết kế tổng quát → Thiết kế chi tiết. Đây là khung làm việc sẽ được dùng xuyên suốt giáo trình.
- **Ngôn ngữ và công cụ:** Java được chọn làm ngôn ngữ minh họa chính vì thuần OOP, cú pháp chặt chẽ, hệ sinh thái phong phú và nhu cầu thị trường cao. Eclipse, NetBeans, IntelliJ IDEA và VS Code là các IDE phổ biến cho lập trình Java.

Câu hỏi ôn tập

1. Phân biệt ba phương pháp tiếp cận trong lập trình. Mỗi phương pháp phù hợp với quy mô chương trình như thế nào?
2. Mối quan hệ giữa lớp và đối tượng là gì? Hãy dùng phép so sánh thực tế để minh họa.
3. Giải thích nguyên lý bao bọc và che giấu thông tin. Tại sao việc cho phép truy cập trực tiếp vào dữ liệu bên trong đối tượng là một thiết kế không tốt?
4. Quan hệ *is-a* trong kế thừa khác gì với quan hệ *has-a*? Cho ví dụ minh họa.
5. Đa hình mang lại lợi ích gì khi lập trình ở mức tổng quát?
6. Liệt kê sáu bước trong quy trình phân tích và thiết kế hướng đối tượng. Tại sao không nên bỏ qua bước đặc tả yêu cầu?
7. So sánh Java và Python dưới góc độ dạy và học OOP. Tại sao giáo trình này chọn Java?

Bài tập thực hành

1. **(Nhận biết)** Từ bản mô tả sau, hãy gạch chân các danh từ và xác định các lớp ứng viên: *"Một hiệu sách cần hệ thống quản lý đơn hàng trực tuyến. Khách hàng có thể duyệt danh mục sách, thêm sách vào giỏ hàng, và đặt hàng. Mỗi đơn hàng gồm một hoặc nhiều đầu sách với số lượng tương ứng."*
2. **(Thay đổi có kiểm soát)** Quay lại ví dụ hệ thống quản lý thư viện. Nếu thư viện bổ sung loại tài liệu mới là "báo và tạp chí" (chỉ đọc tại chỗ, không được mượn về), thiết kế lớp của ta cần thay đổi như thế nào?
3. **(Vận dụng)** Áp dụng **ba bước đầu tiên** của quy trình thiết kế OOP cho bài toán: *"Xây dựng ứng dụng đặt vé xem phim."* Yêu cầu: (a) viết mô tả bài toán trong 3–5 câu, (b) liệt kê ít nhất 4 yêu cầu chức năng, (c) xác định 4–5 lớp chính kèm lý do chọn.

Câu hỏi thảo luận

1. Trong thực tế, nhiều lập trình viên bỏ qua bước thiết kế và "nhảy thẳng vào code". Theo bạn, cách làm đó có thể gây ra những vấn đề gì khi dự án phát triển đến quy mô vài nghìn dòng? Khi nào thì việc bỏ qua thiết kế là chấp nhận được?
2. Một số ngôn ngữ như Python và JavaScript hỗ trợ OOP nhưng không ép buộc

người dùng phải tổ chức code theo lớp. Liệu việc một ngôn ngữ "dễ dãi" hơn có khiến người mới học khó nắm bắt các nguyên lý OOP hơn so với một ngôn ngữ "chặt chẽ" như Java không? Tại sao?

3. Sáu khái niệm OOP được giới thiệu trong chương này — theo bạn, khái niệm nào là khó nắm bắt nhất đối với người mới học? Hãy đề xuất một cách giải thích khác (bằng hình ảnh, phép so sánh, hoặc hoạt động nhóm) để giúp người học hiểu khái niệm đó tốt hơn.