

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



BÀI GIẢNG
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

NGUYỄN MẠNH SƠN

HÀ NỘI 2020

MỤC LỤC

MỤC LỤC	2
GIỚI THIỆU	7
PHẦN 1: NHỮNG KHÁI NIỆM CƠ BẢN	9
CHƯƠNG 1: TỔNG QUAN VỀ CÁCH TIẾP CẬN HƯỚNG ĐỐI TƯỢNG	10
1.1 PHƯƠNG PHÁP TIẾP CẬN CỦA LẬP TRÌNH TRUYỀN THỐNG	10
1.1.1 Lập trình tuyến tính	10
1.1.2 Lập trình cấu trúc	11
1.2 PHƯƠNG PHÁP TIẾP CẬN HƯỚNG ĐỐI TƯỢNG	13
1.2.1 Phương pháp lập trình hướng đối tượng	13
1.2.2 Phương pháp phân tích và thiết kế hướng đối tượng	15
1.3 SO SÁNH HAI CÁCH TIẾP CẬN	17
1.4 XU HƯỚNG PHÁT TRIỂN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	18
TỔNG KẾT CHƯƠNG 1	20
CHƯƠNG 2: NHỮNG KHÁI NIỆM CƠ BẢN CỦA LTHDT	21
2.1 CÁC KHÁI NIỆM CƠ BẢN	21
2.1.1 Đối tượng	21
2.1.2 Lớp đối tượng	22
2.1.3 Trừu tượng hoá đối tượng theo chức năng	24
2.1.4 Trừu tượng hoá đối tượng theo dữ liệu	25
2.1.5 Khái niệm kế thừa	26
2.1.6 Khái niệm đóng gói	28
2.1.7 Khái niệm đa hình	29
2.2 SO SÁNH LỚP VÀ CẤU TRÚC	30
2.3 THÀNH PHẦN PRIVATE VÀ PUBLIC CỦA LỚP	31
2.4 MỘT SỐ NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	32
2.4.1 C++	32
2.4.2 ASP.NET và C#.NET	33
2.4.3 Java	33
TỔNG KẾT CHƯƠNG 2	34
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2	35
PHẦN 2: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA	36

CHƯƠNG 3: GIỚI THIỆU VỀ JAVA.....	37
3.1 LỊCH SỬ PHÁT TRIỂN CỦA JAVA.....	37
3.1.1 Java.....	37
3.1.2 Đặc trưng của ngôn ngữ Java	37
3.1.3 Cài đặt Java.....	40
3.2 KIẾN TRÚC CHƯƠNG TRÌNH XÂY DỰNG TRÊN JAVA	41
3.2.1 Kiến trúc chương trình Java.....	41
3.2.2 Chương trình Java đầu tiên.....	44
3.2.3 Phân tích chương trình đầu tiên.....	45
3.3 CÁC KIỂU DỮ LIỆU VÀ TOÁN TỬ CƠ BẢN TRÊN JAVA	47
3.3.1 Khai báo biến.....	47
3.3.2 Kiểu dữ liệu	47
3.3.3 Các toán tử.....	49
3.4 CÁC CẤU TRÚC LỆNH TRÊN JAVA.....	53
3.4.1 Câu lệnh if-else.....	53
3.4.2 Câu lệnh switch-case	54
3.4.3 Vòng lặp While.....	56
3.4.4 Vòng lặp do-while	57
3.4.5 Vòng lặp for.....	58
3.5 CASE STUDY I	59
TỔNG KẾT CHƯƠNG 3	61
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3	62
CHƯƠNG 4: KẾ THỪA VÀ ĐA HÌNH TRÊN JAVA	65
4.1 KẾ THỪA ĐƠN	65
4.1.1 Lớp.....	65
4.1.2 Sự kế thừa	70
4.2 KẾ THỪA BỘI.....	73
4.2.1 Giao tiếp	73
4.2.2 Sử dụng giao tiếp.....	74
4.3 LỚP TRỪU TƯỢNG	76
4.3.1 Khai báo.....	76
4.3.2 Sử dụng lớp trừu tượng.....	78
4.4 ĐA HÌNH	79
4.4.1 Nạp chồng.....	80
4.4.2 Đa hình.....	80

4.5 CASE STUDY II.....	82
4.5.1 Lớp Human.....	82
4.5.2 Lớp Person.....	83
4.5.3 Lớp Employee	84
4.5.4 Chương trình demo.....	86
TỔNG KẾT CHƯƠNG 4.....	87
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4	88
CHƯƠNG 5: BIỂU DIỄN VÀ CÀI ĐẶT CÁC CTDL TRÊN JAVA.....	93
5.1 PHƯƠNG PHÁP DUYỆT VÀ ĐỆ QUI	93
5.1.1 Các phương pháp duyệt.....	93
5.1.2 Phương pháp đệ qui	94
5.2 PHƯƠNG PHÁP SẮP XẾP VÀ TÌM KIẾM.....	94
5.2.1 Các phương pháp sắp xếp.....	94
5.2.2 Các phương pháp tìm kiếm.....	96
5.3 NGĂN XẾP VÀ HÀNG ĐỢI.....	98
5.3.1 Ngăn xếp.....	98
5.3.2 Hàng đợi	100
5.4 DANH SÁCH LIÊN KẾT	102
5.4.1 Danh sách liên kết đơn	102
5.4.2 Danh sách liên kết kép.....	108
5.5 CÂY NHỊ PHÂN.....	113
5.6 ĐỒ THỊ.....	119
5.6.1 Biểu diễn đồ thị.....	119
5.6.2 Cài đặt đồ thị không có trọng số.....	121
5.6.3 Cài đặt đồ thị có trọng số.....	126
5.7 CASE STUDY III.....	131
TỔNG KẾT CHƯƠNG 5	137
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5	137
CHƯƠNG 6: LẬP TRÌNH GIAO DIỆN TRÊN JAVA.....	139
6.1 GIAO DIỆN VỚI CÁC ĐỐI TƯỢNG CƠ BẢN	139
6.1.1 Các đối tượng container cơ bản	139
6.1.2 Các đối tượng component cơ bản	142
6.1.3 Các sự kiện cơ bản của đối tượng.....	146
6.2 GIAO DIỆN VỚI CÁC ĐỐI TƯỢNG MULTIMEDIA	150
6.2.1 Ô đánh dấu và nút chọn	150

6.2.2 Lựa chọn	152
6.2.3 Danh sách	154
6.2.4 Trình đơn	157
6.3 CÁC KỸ THUẬT TẠO TABLES	160
6.3.1 Trình bày Flow Layout	161
6.3.2 Trình bày Grid Layout	162
6.3.3 Trình bày Border Layout	163
6.3.4 Trình bày GridBag Layout	164
6.3.5 Trình bày Null Layout	167
6.4 HTML & APPLET	168
6.4.1 Cấu trúc của một Applet	168
6.4.2 Sử dụng applet	170
6.4.3 Truyền tham số cho Applet	173
6.5 GIỚI THIỆU VỀ SWING	174
6.5.1 Mở rộng các đối tượng component	175
6.5.2 Mở rộng các đối tượng container	176
6.6 CASE STUDY IV	179
TỔNG KẾT CHƯƠNG 6	186
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 6	187
CHƯƠNG 7: LẬP TRÌNH GENERIC.....	188
7.1 Giới thiệu về Generic:	189
7.1.1 Đặt vấn đề	189
7.1.2. Khái niệm Generic	189
7.2 Các phương thức Generic trong Java:	190
7.3 Các lớp Generic Classes	192
7.4 Các kí tự đại diện Generics	195
7.5 Ưu và một số lưu ý khi sử dụng Gerenics.	197
7.5.1 Ưu điểm	197
7.5.2 Một số lưu ý khi sử dụng Gerenics:	197
CHƯƠNG 8: THƯ VIỆN CÁC COLLECTION TRONG JAVA VÀ ÁP DỤNG.	199
8.1 Các thành phần của Collection	200
8.2 Giới thiệu Collection:	201
8.3 Duyệt Collection:	202
8.3.1 Sử dụng for – each:	202
8.3.2 Sử dụng Iterator:	202

8.2.3 Thao tác số lượng lớn (Bulk)	203
8.2.4 Thao tác mảng (array):	205
8.3 Giới thiệu List và cách sử dụng	206
8.3.1 So sánh với Vector	207
8.3.2 ArrayList	208
8.3.3 Danh sách liên kết (Linked List)	208
8.3.4 Giới thiệu Queue:	211
8.4 Giới thiệu Set Interface:	216
8.5 Giới thiệu về Map	223
TÀI LIỆU THAM KHẢO	230
PHỤ LỤC: BÀI TẬP LẬP TRÌNH JAVA	231

GIỚI THIỆU

Trong những năm gần đây, lập trình hướng đối tượng đã trở nên gần gũi nhờ sự ra đời liên tiếp của các ngôn ngữ lập trình hướng đối tượng. Sức mạnh của phương pháp lập trình hướng đối tượng thể hiện ở chỗ khả năng mô hình hoá hệ thống dựa trên các đối tượng thực tế, khả năng đóng gói và bảo vệ an toàn dữ liệu, khả năng sử dụng lại mã nguồn để tiết kiệm chi phí và tài nguyên; đặc biệt là khả năng chia sẻ mã nguồn trong cộng đồng lập trình viên chuyên nghiệp. Những điểm mạnh này hứa hẹn sẽ thúc đẩy phát triển một môi trường lập trình tiên tiến cùng với nền công nghiệp lắp ráp phần mềm với các thư viện thành phần có sẵn.

Tài liệu này nhằm giới thiệu cho các sinh viên một cái nhìn tổng quan về phương pháp lập trình hướng đối tượng cùng cung cấp những kiến thức, các kỹ thuật cơ bản cho phát triển các ứng dụng của mình dựa trên ngôn ngữ lập trình Java - một trong những ngôn ngữ lập trình hướng đối tượng thông dụng nhất hiện nay.

Nội dung của tài liệu này bao gồm hai phần chính:

Phần thứ nhất trình bày những khái niệm và các vấn đề cơ bản của lập trình hướng đối tượng bao gồm tổng quan về cách tiếp cận hướng đối tượng và các khái niệm đối tượng, lớp, kế thừa, đóng gói, đa hình...

Phần thứ hai trình bày chi tiết phương pháp lập trình hướng đối tượng với ngôn ngữ lập trình Java.

Nội dung của tài liệu bao gồm 7 chương:

Chương 1: Tổng quan về cách tiếp cận hướng đối tượng. Trình bày sự tiến hoá của cách tiếp cận từ lập trình truyền thống đến cách tiếp cận của lập trình hướng đối tượng và xu hướng phát triển của lập trình hướng đối tượng hiện nay.

Chương 2: Những khái niệm cơ bản của lập trình hướng đối tượng. Trình bày các khái niệm cơ bản như: đối tượng, lớp đối tượng với các thuộc tính và phương thức, tính kế thừa và đa hình, tính đóng gói của lập trình hướng đối tượng. Chương này cũng giới thiệu tổng quan một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay.

Chương 3: Ngôn ngữ Java. Giới thiệu những khái niệm và những quy ước ban đầu của ngôn ngữ lập trình Java: Cấu trúc chương trình, cách biên dịch, cách đặt tên biến, kiểu dữ liệu, các toán tử và cấu trúc lệnh của ngôn ngữ Java.

Chương 4: Kế thừa và đa hình trên Java. Trình bày các kỹ thuật lập trình hướng đối tượng dựa trên ngôn ngữ Java: Khai báo lớp, các thuộc tính và phương thức của lớp; kỹ thuật thừa kế, các lớp trừu tượng, cài đặt nạp chồng và đa hình trên Java.

Chương 5: Biểu diễn và cài đặt các cấu trúc dữ liệu trừu tượng trên Java. Trình bày kỹ thuật cài đặt và sử dụng một số cấu trúc dữ liệu quen thuộc trong Java: ngăn xếp, hàng đợi, danh sách liên kết, cây nhị phân và đồ thị.

Chương 6: Lập trình giao diện trên Java. Trình bày các kỹ thuật lập trình giao diện trên Java: Lập trình với các giao diện cơ bản trong thư viện AWT, lập trình giao diện với Applet và HTML, lập trình giao diện nâng cao với thư viện SWING.

Chương 7: Lập trình Generic. Trình bày về lập trình khái quát (Generic), cách dùng và các trường hợp áp dụng.

Chương 8: Thư viện các Collection và áp dụng: Giới thiệu các lớp trong gói thư viện Collection của Java, cấu trúc, các phương thức và cách sử dụng.

Tài liệu này được viết nhằm phục vụ môn học “**Lập trình hướng đối tượng**” giảng dạy tiếp theo sau môn học Ngôn ngữ lập trình C++ và như vậy khi học môn học này sinh viên sẽ dễ nắm bắt được những đặc trưng khác biệt của ngôn ngữ Java so với C++.

Mặc dù các tác giả đã có nhiều cố gắng trong quá trình biên soạn tài liệu này, song không thể tránh khỏi những thiếu sót. Rất mong nhận được sự đóng góp ý kiến của sinh viên và các bạn đồng nghiệp.

PHẦN 1

NHỮNG KHÁI NIỆM CƠ BẢN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 1

TỔNG QUAN VỀ CÁCH TIẾP CẬN HƯỚNG ĐỐI TƯỢNG

Nội dung chương này nhằm giới thiệu một cách tổng quan về cách tiếp cận hướng đối tượng. Nội dung trình bày bao gồm:

Giới thiệu về cách tiếp cận của lập trình truyền thống.

Giới thiệu cách tiếp cận của lập trình hướng đối tượng.

So sánh sự khác biệt giữa hai cách tiếp cận này.

Xu hướng hiện nay của lập trình hướng đối tượng

1.1 PHƯƠNG PHÁP TIẾP CẬN CỦA LẬP TRÌNH TRUYỀN THỐNG

Lập trình truyền thống đã trải qua hai giai đoạn:

Giai đoạn sơ khai, khi khái niệm lập trình mới ra đời, là lập trình tuyến tính.

Giai đoạn tiếp theo, là lập trình hướng cấu trúc.

1.1.1 Lập trình tuyến tính

Đặc trưng cơ bản của lập trình tuyến tính là tư duy theo lối tuần tự. Chương trình sẽ được thực hiện tuần tự từ đầu đến cuối, lệnh này kế tiếp lệnh kia cho đến khi kết thúc chương trình.

Đặc trưng

Lập trình tuyến tính có hai đặc trưng:

Đơn giản: chương trình được tiến hành đơn giản theo lối tuần tự, không phức tạp.

Đơn luồng: chỉ có một luồng công việc duy nhất, và các công việc được thực hiện tuần tự trong luồng đó.

Tính chất

Ưu điểm: Do tính đơn giản, lập trình tuyến tính có ưu điểm là chương trình đơn giản, dễ hiểu. Lập trình tuyến tính được ứng dụng cho các chương trình đơn giản.

Nhược điểm: Với các ứng dụng phức tạp, người ta không thể dùng lập trình tuyến tính để giải quyết.

Ngày nay, lập trình tuyến tính chỉ tồn tại trong phạm vi các modul nhỏ nhất của các phương pháp lập trình khác. Ví dụ trong một chương trình con của lập trình cấu trúc, các lệnh cũng được thực hiện theo tuần tự từ đầu đến cuối chương trình con.

1.1.2 Lập trình cấu trúc

Trong lập trình hướng cấu trúc, chương trình chính được chia nhỏ thành các chương trình con và mỗi chương trình con thực hiện một công việc xác định. Chương trình chính sẽ gọi đến chương trình con theo một giải thuật, hoặc một cấu trúc được xác định trong chương trình chính.

Các ngôn ngữ lập trình cấu trúc phổ biến là Pascal, C và C++. Riêng C++ ngoài việc có đặc trưng của lập trình cấu trúc do kế thừa từ C, còn có đặc trưng của lập trình hướng đối tượng. Cho nên C++ còn được gọi là ngôn ngữ lập trình nửa cấu trúc, nửa hướng đối tượng.

Đặc trưng

Đặc trưng cơ bản nhất của lập trình cấu trúc thể hiện ở mối quan hệ:

Chương trình = Cấu trúc dữ liệu + Giải thuật

Trong đó:

Cấu trúc dữ liệu là cách tổ chức dữ liệu, cách mô tả bài toán dưới dạng ngôn ngữ lập trình

Giải thuật là một quy trình để thực hiện một công việc xác định

Trong chương trình, giải thuật có quan hệ phụ thuộc vào cấu trúc dữ liệu:

Một cấu trúc dữ liệu chỉ phù hợp với một số hạn chế các giải thuật.

Nếu thay đổi cấu trúc dữ liệu thì phải thay đổi giải thuật cho phù hợp.

Một giải thuật thường phải đi kèm với một cấu trúc dữ liệu nhất định.

Tính chất

Mỗi chương trình con có thể được gọi thực hiện nhiều lần trong một chương trình chính.

Các chương trình con có thể được gọi đến để thực hiện theo một thứ tự bất kì, tùy thuộc vào giải thuật trong chương trình chính mà không phụ thuộc vào thứ tự khai báo của các chương trình con.

Các ngôn ngữ lập trình cấu trúc cung cấp một số cấu trúc lệnh điều khiển chương trình.

Ưu điểm

Chương trình sáng sủa, dễ hiểu, dễ theo dõi.

Tư duy giải thuật rõ ràng.

Nhược điểm

Lập trình cấu trúc không hỗ trợ việc sử dụng lại mã nguồn: Giải thuật luôn phụ thuộc chặt chẽ vào cấu trúc dữ liệu, do đó, khi thay đổi cấu trúc dữ liệu, phải thay đổi giải thuật, nghĩa là phải viết lại chương trình.

Không phù hợp với các phần mềm lớn: tư duy cấu trúc với các giải thuật chỉ phù hợp với các bài toán nhỏ, nằm trong phạm vi một modul của chương trình. Với dự án phần mềm lớn, lập trình cấu trúc tỏ ra không hiệu quả trong việc giải quyết mối quan hệ vĩ mô giữa các modul của phần mềm.

Vấn đề

Vấn đề cơ bản của lập trình cấu trúc là bằng cách nào để phân chia chương trình chính thành các chương trình con cho phù hợp với yêu cầu, chức năng và mục đích của mỗi bài toán.

Thông thường, để phân rã bài toán trong lập trình cấu trúc, người ta sử dụng phương pháp thiết kế trên xuống (top-down).

Phương pháp thiết kế trên xuống (top-down)

Phương pháp thiết kế top-down tiếp cận bài toán theo hướng từ trên xuống dưới, từ tổng quan đến chi tiết. Theo đó, một bài toán được chia thành các bài toán con nhỏ hơn. Mỗi bài toán con lại được chia nhỏ tiếp, nếu có thể, thành các bài toán con nhỏ hơn nữa.

Quá trình này còn được gọi là quá trình làm mịn dần. Quá trình làm mịn dần sẽ dừng lại khi các bài toán con không cần chia nhỏ thêm nữa. Nghĩa là khi mỗi bài toán con đều có thể giải quyết bằng một chương trình con với một giải thuật đơn giản.

Ví dụ, sử dụng phương pháp top-down để giải quyết bài toán là xây một căn nhà mới. Khi đó, ta có thể phân rã bài toán theo các bước như sau:

Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như: làm móng, đổ cột, đổ trần, xây tường, lợp mái.

Ở mức thứ hai, phân rã các công việc ở mức thứ nhất: việc làm móng nhà có thể phân rã tiếp thành các công việc: đào móng, gia cố nền, làm khung sắt, đổ bê tông. Công việc đổ cột được phân rã thành ...

Ở mức thứ ba, phân rã các công việc của mức thứ hai: việc đào móng có thể phân chia tiếp thành các công việc: đo đạc, cắm mốc, căng dây, đào và kiểm tra móng. Việc gia cố nền được phân rã thành ...

Quá trình phân rã có thể dừng ở mức này, bởi vì các công việc con thu được là: đo đạc, cắm mốc, căng dây, đào... có thể thực hiện được ngay, không cần chia nhỏ thêm nữa.

Lưu ý:

Cùng sử dụng phương pháp top-down với cùng một bài toán, nhưng có thể cho ra nhiều kết quả khác nhau. Nguyên nhân là do sự khác nhau trong tiêu chí để phân rã một bài toán thành các bài toán con.

Ví dụ, vẫn áp dụng phương pháp top-down để giải quyết bài toán xây nhà, nhưng nếu sử dụng một cách khác để phân chia bài toán, ta có thể thu được kết quả khác biệt so với phương pháp ban đầu:

Ở mức thứ nhất, chia bài toán xây nhà thành các bài toán nhỏ hơn như: làm phần gỗ, làm phần sắt, làm phần bê tông và làm phần gạch.

Ở mức thứ hai, phân rã các công việc ở mức thứ nhất: việc làm gỗ có thể chia thành các công việc như: xẻ gỗ, gia công gỗ, tạo khung, lắp vào nhà. Việc làm sắt có thể chia nhỏ thành...

Rõ ràng, với cách làm mịn thế này, ta sẽ thu được một kết quả khác hẳn với cách thức đã thực hiện ở phần trên.

1.2 PHƯƠNG PHÁP TIẾP CẬN HƯỚNG ĐỐI TƯỢNG

1.2.1 Phương pháp lập trình hướng đối tượng

Xuất phát từ hai hạn chế chính của phương pháp lập trình cấu trúc:

Không quản lý được sự thay đổi dữ liệu khi có nhiều chương trình cùng thay đổi một biến chung. Vấn đề này đặc biệt nghiêm trọng khi các ứng dụng ngày càng lớn, người ta không thể kiểm soát được sự truy nhập đến các biến dữ liệu chung.

Không tiết kiệm được tài nguyên con người: Giải thuật gắn liền với cấu trúc dữ liệu, nếu thay đổi cấu trúc dữ liệu, sẽ phải thay đổi giải thuật, và do đó, phải viết lại mã chương trình từ đầu.

Để khắc phục được hai hạn chế này khi giải quyết các bài toán lớn, người ta xây dựng một phương pháp tiếp cận mới, là phương pháp lập trình hướng đối tượng, với hai mục đích chính:

Đóng gói dữ liệu để hạn chế sự truy nhập tự do vào dữ liệu, không quản lý được.

Cho phép sử dụng lại mã nguồn, hạn chế việc phải viết lại mã từ đầu cho các chương trình.

Việc đóng gói dữ liệu được thực hiện theo phương pháp trừu tượng hoá đối tượng thành lớp từ thấp lên cao như sau:

Thu thập các thuộc tính của mỗi đối tượng, gắn các thuộc tính vào đối tượng tương ứng.

Nhóm các đối tượng có các thuộc tính tương tự nhau thành nhóm, loại bỏ bớt các thuộc tính cá biệt, chỉ giữ lại các thuộc tính chung nhất. Đây được gọi là quá trình trừu tượng hoá đối tượng thành lớp.

Đóng gói dữ liệu của các đối tượng vào lớp tương ứng. Mỗi thuộc tính của đối tượng trở thành một thuộc tính của lớp tương ứng.

Việc truy nhập dữ liệu được thực hiện thông qua các phương thức được trang bị cho lớp. Không được truy nhập tự do trực tiếp đến dữ liệu.

Khi có thay đổi trong dữ liệu của đối tượng, ta chỉ cần thay đổi các phương thức truy nhập thuộc tính của lớp, mà không cần phải thay đổi mã nguồn của các chương trình sử dụng lớp tương ứng.

Việc cho phép sử dụng lại mã nguồn được thực hiện thông qua cơ chế kế thừa trong lập trình hướng đối tượng. Theo đó:

Các lớp có thể được kế thừa nhau để tận dụng các thuộc tính, các phương thức của nhau.

Trong lớp dẫn xuất (lớp được kế thừa) có thể sử dụng lại các phương thức của lớp cơ sở (lớp bị lớp khác kế thừa) mà không cần thiết phải cài đặt lại mã nguồn.

Ngay cả khi lớp dẫn xuất định nghĩa lại các phương thức cho mình, lớp cơ sở cũng không bị ảnh hưởng và không phải sửa lại bất kì một đoạn mã nguồn nào.

Ngôn ngữ lập trình hướng đối tượng phổ biến hiện nay là Java và C++. Tuy nhiên, C++ mặc dù cũng có những đặc trưng cơ bản của lập trình hướng đối tượng nhưng vẫn không phải là ngôn ngữ lập trình thuần hướng đối tượng. Java thật sự là một ngôn ngữ lập trình thuần hướng đối tượng.

Đặc trưng

Lập trình hướng đối tượng có hai đặc trưng cơ bản:

Đóng gói dữ liệu: dữ liệu luôn được tổ chức thành các thuộc tính của lớp đối tượng.

Việc truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp.

Sử dụng lại mã nguồn: việc sử dụng lại mã nguồn được thể hiện thông qua cơ chế kế thừa. Cơ chế này cho phép các lớp đối tượng có thể kế thừa từ các lớp đối tượng khác. Khi đó, trong các lớp kế thừa, có thể sử dụng các phương thức (mã nguồn) của các lớp bị kế thừa, mà không cần phải định nghĩa lại.

Ưu điểm

Lập trình hướng đối tượng có một số ưu điểm nổi bật:

Không còn nguy cơ dữ liệu bị thay đổi tự do trong chương trình. Vì dữ liệu đã được đóng gói vào các đối tượng. Nếu muốn truy nhập vào dữ liệu phải thông qua các phương thức cho phép của đối tượng.

Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi các đối mã nguồn của các đối tượng khác, mà chỉ cần thay đổi một số hàm thành phần của đối tượng bị thay đổi. Điều này hạn chế sự ảnh hưởng xấu của việc thay đổi dữ liệu đến các đối tượng khác trong chương trình.

Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên. Vì nguyên tắc kế thừa cho phép các lớp kế thừa sử dụng các phương thức được kế thừa từ lớp khác như những phương thức của chính nó, mà không cần thiết phải định nghĩa lại.

Phù hợp với các dự án phần mềm lớn, phức tạp.

1.2.2 Phương pháp phân tích và thiết kế hướng đối tượng

Một vấn đề cơ bản đặt ra cho phương pháp hướng đối tượng là từ một bài toán ban đầu, làm sao để thu được một tập các đối tượng, với các chức năng được phối hợp với nhau, đáp ứng được yêu cầu của bài toán đặt ra?

Phương pháp phân tích thiết kế hướng đối tượng ra đời nhằm trả lời cho câu hỏi này. Mục đích là xây dựng một tập các lớp đối tượng tương ứng với mỗi bài toán, phương pháp này tiến hành theo hai pha chính:

Pha phân tích: Chuyển đổi yêu cầu bài toán từ ngôn ngữ tự nhiên sang ngôn ngữ mô hình.

Pha thiết kế: Chuyển đổi đặc tả bài toán dưới dạng ngôn ngữ mô hình sang một mô hình cụ thể có thể cài đặt được.

Hai pha phân tích và thiết kế này bao gồm nhiều bước khác nhau:

Mô tả bài toán

Đặc tả yêu cầu

Trích chọn đối tượng

Mô hình hoá lớp đối tượng

Thiết kế tổng quan

Thiết kế chi tiết.

Bước 1: Mô tả bài toán

Bài toán ban đầu được phát biểu dưới dạng ngôn ngữ tự nhiên, bao gồm:

Mục đích, chức năng chung

Các yêu cầu về thông tin dữ liệu

Các yêu cầu về chức năng thực hiện

Bước 2: Đặc tả yêu cầu

Các yêu cầu được hình thức hoá lên một mức cao hơn bằng cách sử dụng ngôn ngữ kịch bản (scenario) để mô tả. Trong một kịch bản, mỗi chức năng, mỗi hoạt động được mô tả bằng một kịch bản, bao gồm:

Các tác nhân tham gia vào kịch bản.

Vai trò của mỗi tác nhân trong kịch bản.

Thứ tự các hành động mà mỗi tác nhân thực hiện: khi nào thực hiện, tác động vào tác nhân nào, thông tin nào được trao đổi.

Quá trình trên được tiến hành với tất cả các chức năng yêu cầu của hệ thống.

Bước 3: Trích chọn đối tượng

Bước này sẽ tiến hành đề xuất các đối tượng có thể có mặt trong hệ thống:

Dựa vào các kịch bản được mô tả trong bước hai, chọn ra các tác nhân có xuất hiện để đề xuất thành các đối tượng.

Lựa chọn các đối tượng bằng cách loại bỏ các tác nhân bên ngoài hệ thống, các tác nhân trùng lặp.

Cuối cùng, ta thu được tập các đối tượng của hệ thống.

Bước 4: Mô hình hoá lớp đối tượng

Bước này tiến hành trừu tượng hoá đối tượng thành các lớp:

Thu thập tất cả các thuộc tính của mỗi đối tượng vừa thu thập được, dựa vào yêu cầu về thông tin trong yêu cầu hệ thống (từ bước 1).

Thu thập các hành động mà mỗi đối tượng cần thực hiện, dựa vào các kịch bản mà đối tượng tương ứng có tham gia (trong bước 2).

Nhóm các đối tượng tương tự nhau, hoặc có nhiều thuộc tính gần giống nhau.

Loại bỏ một số thuộc tính cá biệt, riêng tư của một số đối tượng trong nhóm.

Mô hình mỗi nhóm đối tượng còn lại thành lớp: Các thuộc tính chung của các đối tượng thành thuộc tính của lớp, các hành động của các đối tượng thành phương thức của lớp.

Kết quả thu được một tập các lớp đối tượng ban đầu của hệ thống.

Bước 5: Thiết kế tổng quát

Bước này sẽ tiến hành thiết kế vĩ mô, nghĩa là thiết kế mối quan hệ giữa các lớp trong hệ thống:

Xác định sơ đồ thừa kế, nếu có, giữa các lớp: Nếu hai lớp có một số thuộc tính chung, thì tách các thuộc tính chung làm thành một lớp cơ sở, và hai lớp ban đầu đều dẫn xuất từ lớp cơ sở đó. Thông thường, lớp các trừu tượng (chung nhất) sẽ làm lớp cơ sở, lớp càng cụ thể, càng chi tiết thì làm lớp dẫn xuất (lớp con, cháu).

Xác định tương tác, nếu có, giữa các lớp: Dựa vào các kịch bản được mô tả trong bước 2, hai tác nhân có tương tác với nhau thì hai lớp tương ứng ở bước này cũng có tương tác với nhau.

Kết quả thu được của bước này là một sơ đồ quan hệ bên ngoài giữa các lớp trong hệ thống.

Bước 6: Thiết kế chi tiết

Bước này sẽ thực hiện thiết kế ở mức vi mô, nghĩa là thiết kế kiến trúc bên trong của mỗi lớp đối tượng:

Tổ chức dữ liệu của lớp theo các thuộc tính. Qui định phạm vi truy nhập cho từng thuộc tính.

Thiết kế chi tiết cách xử của lớp đối tượng thông qua các phương thức của lớp: Xác định kiểu dữ liệu trả về, kiểu tham số của phương thức, mô tả thuật toán chi tiết cho từng phương thức, nếu cần.

Kết quả thu được của bước này là một tập các lớp với thiết kế chi tiết kiến trúc bên trong.

Sau các bước phân tích thiết kế hướng đối tượng từ một yêu cầu của bài toán ban đầu, ta thu được một mô hình hệ thống hướng đối tượng chi tiết:

Có cái nhìn tổng quan, vĩ mô về hệ thống bằng mô hình thiết kế tổng quan, chỉ rõ số lượng các lớp đối tượng, mối quan hệ kế thừa và quan hệ tương tác giữa các lớp đối tượng trong hệ thống.

Có cái nhìn chi tiết, vi mô về hệ thống bằng mô hình thiết kế chi tiết. Mô hình này chỉ rõ bên trong mỗi lớp đối tượng: các thuộc tính, các phương thức với kiểu trả về và kiểu tham số, thuật toán chi tiết cho mỗi phương thức.

Sau pha phân tích và thiết kế hướng đối tượng, ta thu được đặc tả hệ thống dưới dạng mô hình các lớp: quan hệ giữa các lớp và kiến trúc bên trong của mỗi lớp. Đây sẽ là đầu vào cho pha tiếp theo, pha lập trình hướng đối tượng, như chúng ta đã biết.

1.3 SO SÁNH HAI CÁCH TIẾP CẬN

Phương pháp tiếp cận hướng đối tượng có bản chất hoàn toàn khác với phương pháp tiếp cận truyền thống (phương pháp tiếp cận hướng cấu trúc) trên nhiều mặt:

Phương pháp mô hình bài toán khác nhau.

Đặc trưng khác nhau về đóng gói

Ưu / nhược điểm khác nhau.

Lĩnh vực ứng dụng khác nhau.

Khác nhau về phương pháp mô hình

Hai phương pháp này khác nhau hoàn toàn ở cách tiếp cận và mô hình bài toán, phương pháp hướng đối tượng tiến hành theo phương pháp từ dưới lên trên, từ thấp lên cao, từ cụ thể đến trừu tượng. Trong khi đó, phương pháp cấu trúc tiếp cận theo phương pháp từ trên xuống dưới, từ tổng quan đến chi tiết:

Phương pháp hướng đối tượng bắt đầu bằng những đối tượng cụ thể, tập hợp các thuộc tính của từng đối tượng. Sau đó, nhóm các đối tượng tương tự nhau thành nhóm, loại bỏ các thuộc tính quá cá biệt, chỉ giữ lại các thuộc tính chung nhất, nhóm thành lớp.

Cho nên, quá trình hình thành lớp là quá trình đi từ thấp lên cao, từ cụ thể ở mức thấp đến trừu tượng hoá ở mức cao.

Trong khi đó, phương pháp hướng cấu trúc lại đi theo chiều ngược lại. Phương pháp này bắt đầu từ một bài toán tổng quan, ở mức khái quát cao, chia nhỏ dần và làm mịn dần cho đến khi thu được một tập các bài toán con, nhỏ hơn, cụ thể hơn, chi tiết hơn.

Khác nhau về đặc trưng đóng gói

Hai phương pháp tiếp cận này cũng có những đặc trưng hoàn toàn khác nhau:

Phương pháp hướng đối tượng có đặc trưng là dữ liệu được đóng gói để hạn chế truy nhập tự do trực tiếp vào dữ liệu. Thứ hai là cho phép sử dụng lại mã nguồn để tiết kiệm tài nguyên và công sức lập trình.

Trong khi đó, đặc trưng của phương pháp cấu trúc là cấu trúc dữ liệu và giải thuật và mối quan hệ phụ thuộc chặt chẽ của giải thuật vào cấu trúc dữ liệu.

Khác nhau về ưu nhược điểm

Hai phương pháp này cũng có những ưu nhược điểm trái ngược nhau:

Phương pháp hướng đối tượng có ưu điểm là bảo vệ được dữ liệu tránh bị truy nhập trực tiếp tự do từ bên ngoài, tiết kiệm được tài nguyên và công sức lập trình do có thể dùng lại mã nguồn. Tuy nhiên, phương pháp này lại khá phức tạp, khó theo dõi được luồng dữ liệu và hơn nữa, giải thuật không phải là vấn đề trọng tâm của phương pháp này.

Trái lại, phương pháp hướng cấu trúc lại có ưu điểm là tư duy giải thuật rõ ràng, dễ theo dõi luồng dữ liệu, chương trình đơn giản và dễ hiểu. Tuy nhiên, không bảo vệ được an toàn dữ liệu trong chương trình. Hơn nữa, hạn chế lớn nhất là sự phụ thuộc chặt chẽ của giải thuật vào cấu trúc dữ liệu, khiến cho khi thay đổi cấu trúc dữ liệu, thường phải thay đổi giải thuật, và do đó, phải viết lại mã cho chương trình.

Khác nhau về lĩnh vực áp dụng

Do sự khác nhau về các đặc trưng và sự khác nhau về ưu nhược điểm, cho nên hai phương pháp này cũng có sự khác nhau đáng kể trong lĩnh vực áp dụng:

Phương pháp hướng đối tượng thường được áp dụng cho các bài toán lớn, phức tạp, có nhiều luồng dữ liệu khác nhau, không thể quản lý được bằng phương pháp cấu trúc. Khi đó, người ta dùng phương pháp hướng đối tượng để tận dụng khả năng bảo vệ dữ liệu tránh bị truy nhập tự do. Hơn nữa, tận dụng khả năng dùng lại mã nguồn của phương pháp này để tiết kiệm tài nguyên và công sức.

Trong khi đó, phương pháp cấu trúc thường phù hợp với các bài toán nhỏ, có luồng dữ liệu rõ ràng, cần phải tư duy giải thuật rõ ràng và người lập trình vẫn có khả năng tự quản lý được mọi truy nhập đến các dữ liệu của chương trình.

1.4 XU HƯỚNG PHÁT TRIỂN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng thành phần

Xuất phát từ lập trình hướng đối tượng, tư duy lập trình hướng thành phần (component-oriented programming) theo ý tưởng:

Giải quyết bài toán bằng cách xây dựng một tập các thành phần (component) có tính độc lập tương đối với nhau. Mỗi thành phần đảm nhiệm một phần công việc nhất định.

Sau đó, người ta ghép các thành phần với nhau để thu được một phần mềm thoả mãn một tập các yêu cầu xác định.

Với lập trình hướng thành phần, người ta có thể tiến hành lập trình theo phương pháp sau:

Xây dựng một thư viện các thành phần, mỗi thành phần thực hiện một công việc xác định.

Khi cần phát triển một phần mềm cụ thể, người ta chỉ cần chọn những thành phần có sẵn trong thư viện để ghép lại với nhau. Người lập trình chỉ phải phát triển thêm các thành phần mình cần mà chưa có trong thư viện.

Phương pháp này có những ưu điểm rất lớn:

Lập trình viên có thể chia sẻ với nhau những thành phần mình đã xây dựng cho nhiều người khác dùng chung.

Khi cần, lập trình viên có thể lắp ghép các thành phần có sẵn khác nhau để tạo thành các chương trình có chức năng khác nhau. Tất cả chỉ cần dựa trên công nghệ lắp ghép thành phần, tiết kiệm được rất nhiều công sức lập trình.

Trong xu hướng lập trình hướng thành phần, một số phương pháp lập trình khác đã nảy sinh và đang phát triển mạnh mẽ:

Lập trình hướng agent (agent-oriented programming)

Lập trình hướng aspect (aspect-oriented programming)

Lập trình hướng agent

Lập trình hướng agent có thể xem là một mức trừu tượng cao hơn của lập trình hướng thành phần. Trong đó, các agent là các thành phần có khả năng hoạt động độc lập, tự chủ để hoàn thành công việc của mình. Hơn nữa, các agent có khả năng chủ động liên lạc với các agent khác để có thể phối hợp, cộng tác hay cạnh tranh nhau để hoàn thành nhiệm vụ.

Lập trình hướng agent có hai đặc trưng cơ bản:

Thứ nhất là khả năng tự chủ của mỗi agent để hoàn thành nhiệm vụ riêng của nó.

Thứ hai là tính tổ chức xã hội giữa các agent, cho phép các agent phối hợp, cộng tác, cạnh tranh nhau để hoàn thành nhiệm vụ chung của toàn hệ thống.

Lập trình hướng aspect

Lập trình hướng aspect cũng là một xu hướng của lập trình hướng thành phần. Theo đó, mỗi thành phần có nhiệm vụ hoàn thành theo một luồng công việc hoặc một khía cạnh của vấn đề. Sau đó, tổng hợp các thành phần của các luồng khác nhau, ta thu được giải pháp cho bài toán của mình.

Lập trình hướng aspect có đặc trưng cơ bản:

Tính đóng gói theo luồng công việc, hoặc đóng gói theo khía cạnh của vấn đề.

Tính đơn điệu theo luồng, trong một luồng công việc, các nhiệm vụ được thực hiện liên tiếp nhau, tuần tự như trong lập trình tuyến tính.

TỔNG KẾT CHƯƠNG 1

Nội dung chương 1 đã trình bày các vấn đề tổng quan liên quan đến phương pháp tiếp cận hướng đối tượng trong lập trình:

Các phương pháp tiếp cận truyền thống: lập trình tuyến tính và lập trình cấu trúc.

Phương pháp tiếp cận hướng đối tượng với hai đặc trưng cơ bản: Đóng gói dữ liệu và sử dụng lại mã nguồn.

Lập trình hướng đối tượng, phương pháp phân tích và thiết kế hệ thống hướng đối tượng.

So sánh sự khác biệt của phương pháp hướng đối tượng với các phương pháp truyền thống trên các khía cạnh: Cách tiếp cận bài toán, đặc trưng, ưu nhược điểm và lĩnh vực áp dụng của mỗi phương pháp.

Hiện nay, lập trình hướng thành phần, lập trình hướng agent và lập trình hướng aspect tiến hoá từ lập trình hướng đối tượng đang là xu hướng phát triển mạnh mẽ.

CHƯƠNG 2

NHỮNG KHÁI NIỆM CƠ BẢN CỦA

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nội dung chương này tập trung trình bày các khái niệm cơ bản của lập trình hướng đối tượng:

Đối tượng

Lớp đối tượng

Việc trừu tượng hoá đối tượng theo chức năng

Trừu tượng hoá đối tượng theo dữ liệu

Kế thừa

Đóng gói

Đa hình

Phương pháp cài đặt lớp đối tượng

Giới thiệu một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay.

2.1 CÁC KHÁI NIỆM CƠ BẢN

2.1.1 Đối tượng

Trong lập trình hướng đối tượng, tất cả các thực thể trong hệ thống đều được coi là các đối tượng cụ thể. Đối tượng là một thực thể hoạt động khi chương trình đang chạy.

Ví dụ:

Trong bài toán quản lý buôn bán xe hơi của một cửa hàng kinh doanh, mỗi chiếc xe đang có mặt trong cửa hàng được coi là một đối tượng. Chẳng hạn, một chiếc xe nhãn hiệu “Ford”, màu trắng, giá 5000\$ là một đối tượng.

Trong bài toán quản lý nhân viên của một văn phòng, mỗi nhân viên trong văn phòng được coi là một đối tượng. Chẳng hạn, nhân viên tên là “Vinh”, 25 tuổi làm ở phòng hành chính là một đối tượng.

Một đối tượng là một thực thể đang tồn tại trong hệ thống và được xác định bằng ba yếu tố:

Định danh đối tượng: xác định duy nhất cho mỗi đối tượng trong hệ thống, nhằm phân biệt các đối tượng với nhau.

Trạng thái của đối tượng: là sự tổ hợp của các giá trị của các thuộc tính mà đối tượng đang có.

Hoạt động của đối tượng: là các hành động mà đối tượng có khả năng thực hiện

được.

Trạng thái hiện tại của đối tượng qui định tính chất đặc trưng của đối tượng. Ví dụ, nhân viên trong ví dụ trên có trạng thái là:

Tên là Vinh

Tuổi là 25

Vị trí làm việc là phòng hành chính.

Trong khi đó, trạng thái của chiếc xe trong cửa hàng là:

Nhãn hiệu xe là Ford

Màu sơn xe là trắng

Giá bán xe là 5000\$

Mỗi đối tượng sẽ thực hiện một số hành động. Ví dụ, đối tượng xe hơi có khả năng thực hiện những hành động sau:

Khởi động.

Dừng lại.

Chạy.

Để biểu diễn đối tượng trong lập trình hướng đối tượng, người ta trừu tượng hoá đối tượng để tạo nên khái niệm lớp đối tượng.

2.1.2 Lớp đối tượng

Trong lập trình hướng đối tượng, đối tượng là một thực thể cụ thể, tồn tại trong hệ thống. Trong khi đó, lớp là một khái niệm trừu tượng, dùng để chỉ một tập hợp các đối tượng có mặt trong hệ thống.

Ví dụ:

Trong bài toán quản lý buôn bán xe hơi của một cửa hàng kinh doanh, mỗi chiếc xe đang có mặt trong cửa hàng được coi là một đối tượng. Nhưng khái niệm “Xe hơi” là một lớp đối tượng dùng để chỉ tất cả các loại xe hơi của cửa hàng.

Trong bài toán quản lý nhân viên của một văn phòng, mỗi nhân viên trong văn phòng được coi là một đối tượng. Nhưng khái niệm “Nhân viên” là một lớp đối tượng dùng để chỉ chung chung các nhân viên của văn phòng.

Lưu ý:

Lớp là một khái niệm, mang tính trừu tượng, dùng để biểu diễn một tập các đối tượng.

Đối tượng là một thể hiện cụ thể của lớp, là một thực thể tồn tại trong hệ thống.

Lớp được dùng để biểu diễn đối tượng, cho nên lớp cũng có thuộc tính và phương thức:

Thuộc tính của lớp tương ứng với thuộc tính của các đối tượng.

Phương thức của lớp tương ứng với các hành động của đối tượng.

Ví dụ, lớp xe ô tô được mô tả bằng các thuộc tính và phương thức:

Lớp Xe ô tô

Thuộc tính:

Nhãn hiệu xe

Màu xe

Giá xe

Công suất xe (mã lực)

Phương thức:

Khởi động xe

Chạy xe

Dừng xe

Tắt máy

Lưu ý:

Một lớp có thể có một trong các khả năng sau:

Hoặc chỉ có thuộc tính, không có phương thức.

Hoặc chỉ có phương thức, không có thuộc tính.

Hoặc có cả thuộc tính và phương thức, trường hợp này là phổ biến nhất.

Đặc biệt, lớp không có thuộc tính và phương thức nào là các lớp trừu tượng. Các lớp này không có đối tượng tương ứng.

Lớp và Đối tượng

Lớp và đối tượng, mặc dù có mối liên hệ tương ứng lẫn nhau, nhưng bản chất lại khác nhau:

Lớp là sự trừu tượng hoá của các đối tượng. Trong khi đó, đối tượng là một thể hiện của lớp.

Đối tượng là một thực thể cụ thể, có thực, tồn tại trong hệ thống. Trong khi đó, lớp là một khái niệm trừu tượng, chỉ tồn tại ở dạng khái niệm để mô tả các đặc tính chung của một số đối tượng.

Tất cả các đối tượng thuộc về cùng một lớp có cùng các thuộc tính và các phương thức.

Một lớp là một nguyên mẫu của một đối tượng. Nó xác định các hành động khả thi và các thuộc tính cần thiết cho một nhóm các đối tượng cụ thể.

Nói chung, lớp là khái niệm tồn tại khi phát triển hệ thống, mang tính khái niệm, trừu tượng. Trong khi đó, đối tượng là một thực thể cụ thể tồn tại khi hệ thống đang hoạt động.

2.1.3 Trừu tượng hoá đối tượng theo chức năng

Trừu tượng hoá đối tượng theo chức năng chính là quá trình mô hình hoá phương thức của lớp dựa trên các hành động của các đối tượng. Quá trình này được tiến hành như sau:

Tập hợp tất cả các hành động có thể có của các đối tượng.

Nhóm các đối tượng có các hoạt động tương tự nhau, loại bỏ bớt các hoạt động cá biệt, tạo thành một nhóm chung.

Mỗi nhóm đối tượng đề xuất một lớp tương ứng.

Các hành động chung của nhóm đối tượng sẽ cấu thành các phương thức của lớp tương ứng.

Ví dụ, trong bài toán quản lý cửa hàng bán ô tô. Mỗi ô tô có mặt trong cửa hàng là một đối tượng. Mặc dù mỗi chiếc xe có một số đặc điểm khác nhau về nhãn hiệu, giá xe, màu sắc... nhưng có chung các hành động của một chiếc xe ô tô là:

Có thể khởi động máy.

Có thể chạy.

Có thể dừng lại.

Có thể tắt máy.

Ngoài ra, một số ít xe có thể thực hiện một số hành động cá biệt như:

Có thể giấu đèn pha

Có thể tự bật đèn pha

Có thể tự động phát tín hiệu báo động.

Tuy nhiên, không phải xe nào cũng thực hiện được các hành động này. Cho nên ta loại bỏ các hành động cá biệt của một số xe, chỉ giữ lại các hành động chung nhất, để mô hình thành các phương thức của đối tượng xe ô tô tương ứng với các hành động chung nhất của các xe ô tô.

Lớp Xe ô tô

Phương thức:

Khởi động xe

Chạy xe

Dừng xe

Tắt máy

2.1.4 Trừu tượng hoá đối tượng theo dữ liệu

Trừu tượng hoá đối tượng theo dữ liệu chính là quá trình mô hình hoá các thuộc tính của lớp dựa trên các thuộc tính của các đối tượng tương ứng. Quá trình này được tiến hành như sau:

Tập hợp tất cả các thuộc tính có thể có của các đối tượng.

Nhóm các đối tượng có các thuộc tính tương tự nhau, loại bỏ bớt các thuộc tính cá biệt, tạo thành một nhóm chung.

Mỗi nhóm đối tượng đề xuất một lớp tương ứng.

Các thuộc tính chung của nhóm đối tượng sẽ cấu thành các thuộc tính tương ứng của lớp được đề xuất.

Ví dụ, trong bài toán quản lí cửa hàng bán ô tô. Mỗi ô tô có mặt trong cửa hàng là một đối tượng. Mặc dù mỗi chiếc xe có một số đặc điểm khác nhau về nhãn hiệu, giá xe, màu sắc... nhưng có chung các thuộc tính của một chiếc xe ô tô là:

Các xe đều có nhãn hiệu.

Các xe đều có màu sắc

Các xe đều có giá bán

Các xe đều có công suất động cơ

Ngoài ra, một số ít xe có thể có thêm các thuộc tính:

Có xe có thể có dàn nghe nhạc

Có xe có thể có màn hình xem ti vi

Có xe có lắp kính chống nắng, chống đạn...

Tuy nhiên, đây là các thuộc tính cá biệt của một số đối tượng xe, nên không được đề xuất thành thuộc tính của lớp ô tô. Do đó, ta mô hình lớp ô tô với các thuộc tính chung nhất của các ô tô.

Lớp Xe ô tô

Thuộc tính:

Nhãn hiệu xe

Màu xe

Giá xe

Công suất xe (mã lực)

Ưu điểm của việc trừu tượng hóa

Những ưu điểm của việc trừu tượng hóa là:

Tập trung vào vấn đề cần quan tâm

Xác định những đặc tính thiết yếu và những hành động cần thiết

Giảm thiểu những chi tiết không cần thiết

Việc trừu tượng hóa dữ liệu là cần thiết, bởi vì không thể mô tả tất cả các hành động và các thuộc tính của một thực thể. Vấn đề mấu chốt là tập trung đến những hành vi cốt yếu và áp dụng chúng trong ứng dụng.

2.1.5 Khái niệm kế thừa

Xét trường hợp bài toán quản lý nhân sự và sinh viên của một trường đại học. Khi đó, ta có hai lớp đối tượng chính là lớp Nhân viên và lớp Sinh viên:

<i>Lớp Nhân viên</i>	<i>Lớp Sinh viên</i>
Thuộc tính:	Thuộc tính:
Tên	Tên
Ngày sinh	Ngày sinh
Giới tính	Giới tính
Lương	Lớp
Phương thức:	Phương thức:
Nhập/xem tên	Nhập/xem tên
Nhập/xem ngày sinh	Nhập/xem ngày sinh
Nhập/xem giới tính	Nhập/xem giới tính
Nhập/xem lương	Nhập/xem lớp

Ta nhận thấy rằng hai lớp này có một số thuộc tính và phương thức chung: tên, ngày sinh, giới tính. Tuy nhiên, không thể loại bỏ các thuộc tính cá biệt để gộp chúng thành một lớp duy nhất, vì các thuộc tính lương nhân viên và lớp của sinh viên là cần thiết cho việc quản lý. Vấn đề nảy sinh như sau:

Ta phải viết mã trùng nhau đến hai lần cho các phương thức: nhập/xem tên, nhập/xem ngày sinh, nhập/xem giới tính. Rõ ràng điều này rất tốn công sức.

Nếu khi có sự thay đổi về kiểu dữ liệu, chẳng hạn kiểu ngày sinh được quản lý trong hệ thống, ta phải sửa lại chương trình hai lần.

Để tránh rắc rối do các vấn đề nảy sinh như vậy, lập trình hướng đối tượng sử dụng kỹ thuật kế thừa nhằm nhóm các phần giống nhau của các lớp thành một lớp mới, sau đó cho các lớp ban đầu kế thừa lại lớp được tạo ra. Như vậy, mỗi lớp thừa kế (lớp dẫn xuất, lớp

con) đều có các thuộc tính và phương thức của lớp bị thừa kế (lớp cơ sở, lớp cha).

Quay lại với bài toán quản lí trường đại học, các thuộc tính và phương thức chung giữa lớp Nhân viên và lớp Sinh viên là:

Tên,

Ngày sinh,

Giới tính,

Nhập/xem tên,

Nhập/xem ngày sinh

Nhập/xem giới tính.

Ta tách phần chung này thành một lớp ở mức trừu tượng cao hơn, lớp Người. Lớp Người sẽ làm lớp cha của lớp Nhân viên và lớp Sinh viên. Khi đó, các lớp trở thành:

Lớp Người Thuộc tính: Tên Ngày sinh Giới tính Phương thức: Nhập/xem tên Nhập/xem ngày sinh Nhập/xem giới tính	
Lớp Nhân viên kế thừa từ lớp Người Thuộc tính: Lương Phương thức: Nhập/xem lương	Lớp Sinh viên kế thừa từ lớp Người Thuộc tính: Lớp Phương thức: Nhập/xem lớp

Như vậy, sự kế thừa trong lập trình hướng đối tượng:

Cho phép lớp dẫn xuất có thể sử dụng các thuộc tính và phương thức của lớp cơ sở tương tự như sử dụng các thuộc tính và phương thức của mình.

Cho phép việc chỉ cần cài đặt phương thức ở một lớp cơ sở, mà có thể sử dụng được ở tất cả các lớp dẫn xuất.

Cho phép tránh sự cài đặt trùng lặp mã nguồn của chương trình.

Cho phép chỉ phải thay đổi một lần khi cần phải thay đổi dữ liệu của các lớp.

2.1.6 Khái niệm đóng gói

Xét ví dụ bài toán quản lý nhân viên văn phòng với lớp Nhân viên như sau:

Lớp Nhân viên

Thuộc tính:

Tên

Ngày sinh

Giới tính

Phòng ban

Hệ số lương

Phương thức:

Tính lương nhân viên

Khi đó, cách tính lương cho nhân viên là khác nhau đối với mỗi người:

$$\text{<Tiền lương>} = \text{<Hệ số lương>} * \text{<Lương cơ bản>} * \text{<Tỉ lệ phần trăm>}$$

Trong đó, tỉ lệ phần trăm là khác nhau cho mỗi phòng ban, ví dụ:

Phòng kế hoạch là 105%

Phòng hành chính là 100%

Phòng nhân sự là 110%

Khi đó, tùy vào thuộc tính phòng ban khác nhau mà ta phải dùng công thức tỉ lệ khác nhau để tính lương cho mỗi nhân viên.

Tuy nhiên, cách tính cụ thể này là công việc bên trong của phương thức tính tiền lương của lớp Nhân viên. Với mỗi ứng dụng, khi tạo một đối tượng cụ thể của lớp nhân viên, ta chỉ cần truyền các tham số thuộc tính cho đối tượng, sau đó gọi phương thức tính tiền lương cho đối tượng nhân viên đó, ta sẽ biết được tiền lương của nhân viên. Cách gọi phương thức tính tiền lương là hoàn toàn giống nhau cho tất cả các đối tượng nhân viên của văn phòng.

Sự giống nhau về cách sử dụng phương thức cho các đối tượng của cùng một lớp, mặc dù bên trong phương thức có các cách tính toán khác nhau với các đối tượng khác nhau, được gọi là tính đóng gói dữ liệu của lập trình hướng đối tượng. Như vậy, tính đóng gói dữ liệu của lập trình hướng đối tượng:

Cho phép che dấu sự cài đặt chi tiết bên trong của phương thức. Khi sử dụng chỉ cần gọi các phương thức theo một cách thống nhất, mặc dù các phương thức có thể được cài đặt khác nhau cho các trường hợp khác nhau.

Cho phép che dấu dữ liệu bên trong của đối tượng. Khi sử dụng, ta không biết được thực sự bên trong đối tượng có những gì, ta chỉ thấy được những gì đối tượng cho phép truy nhập vào.

Cho phép hạn chế tối đa việc sửa lại mã chương trình. Khi phải thay đổi công thức tính toán của một phương thức, ta chỉ cần thay đổi mã bên trong của phương thức, mà không phải thay đổi các chương trình gọi đến phương thức bị thay đổi.

2.1.7 Khái niệm đa hình

Trở lại với ví dụ về quản lý trường đại học, với hai lớp Nhân viên và lớp Sinh viên, đều kế thừa từ lớp Người. Khi đó, ta thêm vào mỗi lớp một phương thức show():

Phương thức show của lớp Người sẽ giới thiệu tên và tuổi của người đó.

Phương thức show của lớp Nhân viên sẽ giới thiệu nhân viên đó có tiền lương là bao nhiêu

Phương thức show của lớp Sinh viên sẽ giới thiệu là sinh viên đó đang học ở lớp nào.

Lớp Người Thuộc tính: Tên Ngày sinh Giới tính Phương thức: Nhập/xem tên Nhập/xem ngày sinh Nhập/xem giới tính show	
Lớp Nhân viên kế thừa từ lớp Người Thuộc tính: Lương Phương thức: Nhập/xem lương show	Lớp Sinh viên kế thừa từ lớp Người Thuộc tính: Lớp Phương thức: Nhập/xem lớp show

Khi đó, nếu trong hệ thống có các đối tượng cụ thể tương ứng với ba lớp, thì:

Khi ta gọi hàm show từ đối tượng của lớp Người, sẽ nhận được tên và tuổi của người đó.

Khi ta gọi phương thức show từ đối tượng của lớp Nhân viên, sẽ nhận được số tiền lương của nhân viên đó.

Khi ta gọi phương thức show từ đối tượng của lớp Sinh viên, ta sẽ biết được lớp học của sinh viên đó.

Việc chỉ cần gọi cùng một phương thức, nhưng từ các đối tượng khác nhau, sẽ cho kết quả khác nhau được gọi là tính đa hình trong lập trình hướng đối tượng. Như vậy, tính đa hình trong lập trình hướng đối tượng:

Cho phép các lớp được định nghĩa các phương thức trùng nhau: cùng tên, cùng số lượng và kiểu tham số, cùng kiểu trả về. Việc định nghĩa phương thức trùng nhau của các lớp kế thừa nhau còn được gọi là sự nạp chồng phương thức.

Khi gọi các phương thức trùng tên, dựa vào đối tượng đang gọi mà chương trình sẽ thực hiện phương thức của lớp tương ứng, và do đó, sẽ cho các kết quả khác nhau.

2.2 SO SÁNH LỚP VÀ CẤU TRÚC

Trong phần này, chúng ta sẽ tiến hành so sánh Class (Lớp) và Structure (Cấu trúc) trên nhiều khía cạnh khác nhau:

Mức khái niệm

Mục đích và chức năng

Về ưu và nhược điểm

So sánh ở mức khái niệm

mức khái niệm, Lớp và cấu trúc hoàn toàn khác nhau:

Lớp là khái niệm chỉ có trong lập trình hướng đối tượng; nó được dùng để biểu diễn một tập các đối tượng tương tự nhau.

Trong khi đó, Cấu trúc là khái niệm chỉ tồn tại trong lập trình cấu trúc, không phải là một khái niệm của lập trình hướng đối tượng. Vì trong lập trình hướng đối tượng, tất cả các thực thể đều được coi là một đối tượng, nghĩa là nó phải là một thể hiện cụ thể của một lớp nào đó. Do đó, trong lập trình hướng đối tượng, không có khái niệm Cấu trúc.

So sánh về mục đích và chức năng

Về mục đích, Lớp và Cấu trúc đều có chung một mục đích ban đầu, đó là nhóm một tập hợp các dữ liệu lại với nhau để xử lý đồng bộ và thống nhất: Cấu trúc nhóm các dữ liệu hay phải đi kèm với nhau lại thành một nhóm cho dễ xử lý. Tương tự, Lớp là tập hợp một số thuộc tính chung của đối tượng để xử lý.

Tuy nhiên, Lớp và Cấu trúc cũng có một số khác biệt trên khía cạnh này:

Lớp ngoài mục đích nhóm các thuộc tính dữ liệu của đối tượng, còn nhóm các hoạt động của đối tượng thành các phương thức của Lớp.

Trong khi đó, mặc dù cũng có thể cung cấp các hàm trong Cấu trúc, nhưng mục đích chính của Cấu trúc chỉ là nhóm dữ liệu thành cấu trúc cho dễ xử lý.

So sánh về ưu nhược điểm

Vì có cùng mục đích là nhóm các dữ liệu lại với nhau để xử lý, cho nên Lớp và Cấu trúc có cùng ưu điểm là làm chương trình gọn gàng, xử lý đồng bộ và thống nhất.

Tuy nhiên, Lớp còn có một số ưu điểm mà Cấu trúc không có:

Lớp có khả năng bảo vệ dữ liệu tránh bị truy nhập tự do từ bên ngoài. Các chương trình bên ngoài chỉ có thể truy nhập vào dữ liệu của đối tượng thông qua các phương thức do Lớp cung cấp, không thể tự do truy nhập. Trong khi đó, Cấu trúc mặc dầu đã nhóm dữ liệu với nhau nhưng không có khả năng bảo vệ dữ liệu: Các chương trình bên ngoài vẫn có thể truy nhập tự do vào các thành phần của Cấu trúc.

Lớp có khả năng đóng gói để hạn chế tối đa thay đổi khi phải sửa lại mã chương trình. Khi có sự thay đổi, chỉ cần thay đổi mã của một phương thức, các chương trình bên ngoài sử dụng phương thức đó đều không phải thay đổi. Trong khi đó, nếu thay đổi một thành phần của Cấu trúc, ta phải thay đổi mã của tất cả các chương trình sử dụng thành phần đó của Cấu trúc.

Lớp có thể được kế thừa bởi một Lớp khác, điều này làm tăng khả năng sử dụng lại mã nguồn của chương trình. Trong khi đó, Cấu trúc hoàn toàn không có cơ chế kế thừa, cho nên nhiều khi phải viết lại những đoạn mã giống nhau nhiều lần. Điều này vừa tốn công sức, vừa không an toàn khi có sự thay đổi một trong những đoạn mã giống nhau đó.

2.3 THÀNH PHẦN PRIVATE VÀ PUBLIC CỦA LỚP

Để bảo vệ dữ liệu tránh bị truy nhập tự do từ bên ngoài, lập trình hướng đối tượng sử dụng các từ khoá quy định phạm vi truy nhập các thuộc tính và phương thức của lớp. Một cách tổng quát, lập trình hướng đối tượng chia ra hai mức truy nhập các thành phần lớp:

Private: Truy nhập trong nội bộ lớp.

Protected: Thành phần được bảo vệ, được hạn chế truy nhập như thành phần private (sẽ được trình bày sau).

Public: Truy nhập tự do từ bên ngoài.

Thành phần private

Thành phần private là khu vực dành riêng cho lớp, không chia sẻ với bất kì lớp khác từ bên ngoài. Thành phần private chỉ cho phép truy nhập trong phạm vi nội bộ lớp: Từ

phương thức vào các thuộc tính hoặc giữa các phương thức của lớp với nhau. Các thành phần private không thể truy nhập từ bên ngoài lớp, cũng như từ đối tượng khác.

Trong một lớp, thông thường các thành phần sau sẽ được đặt vào khu vực private của lớp:

Tất cả các thuộc tính dữ liệu của lớp. Các thuộc tính dữ liệu của lớp được đặt vào vùng private nhằm bảo vệ chúng, tránh sự truy nhập tự do từ bên ngoài.

Các phương thức trung gian, được sử dụng như các bước tính toán đệm cho các phương thức khác. Các phương thức trung gian được đặt vào vùng private để thực hiện việc đóng gói trong lập trình hướng đối tượng: Các đối tượng, chương trình bên ngoài không cần, và không thể biết cách tính toán cụ thể bên trong của lớp.

Thành phần public

Thành phần public là khu vực mà Lớp có thể chia sẻ với tất cả các chương trình và đối tượng bên ngoài. Thành phần public có thể được truy nhập từ bên trong lẫn bên ngoài lớp:

Bên trong lớp: từ phương thức lớp vào các thuộc tính dữ liệu của lớp, hoặc giữa các phương thức của lớp với nhau.

Bên ngoài lớp: Từ chương trình bên ngoài hoặc các đối tượng khác vào các phương thức của lớp.

Trong một lớp, thông thường các thành phần sau sẽ được đặt vào vùng chia sẻ public của lớp:

Các phương thức để nhập/xem (set/get) các thuộc tính dữ liệu của lớp. Các phương thức này sẽ cho phép các đối tượng bên ngoài truy nhập vào các thuộc tính dữ liệu của lớp một cách gián tiếp.

Các phương thức cung cấp chức năng hoạt động, cách cư xử của đối tượng đối với môi trường bên ngoài. Các phương thức này thể hiện chức năng của các đối tượng lớp.

2.4 MỘT SỐ NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nội dung phần này sẽ trình bày một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay:

Ngôn ngữ lập trình C++

Ngôn ngữ lập trình ASP.NET và C#.NET

Ngôn ngữ lập trình Java

2.4.1 C++

C++, ra đời vào giữa những năm 1980, là một ngôn ngữ lập trình hướng đối tượng được mở rộng từ ngôn ngữ lập trình cấu trúc C. Cho nên, C++ là ngôn ngữ lập trình nửa hướng đối tượng, nửa hướng cấu trúc.

Những đặc trưng hướng đối tượng của C++

Cho phép định nghĩa lớp đối tượng.

Cho phép đóng gói dữ liệu vào các lớp đối tượng. Cho phép định nghĩa phạm vi truy cập dữ liệu của lớp bằng các từ khoá phạm vi.

Cho phép kế thừa lớp với các kiểu kế thừa khác nhau tùy vào từ khoá dẫn xuất.

Cho phép lớp kế thừa sử dụng các phương thức của lớp bị kế thừa (trong phạm vi quy định).

Cho phép định nghĩa chồng phương thức trong lớp kế thừa.

Những vi phạm hướng đối tượng của C++

Những vi phạm này là do kết quả kế thừa từ ngôn ngữ C, một ngôn ngữ lập trình thuần cấu trúc.

Cho phép định nghĩa và sử dụng các biến dữ liệu tự do.

Cho phép định nghĩa và sử dụng các hàm tự do.

Ngay cả khi dữ liệu được đóng gói vào lớp, dữ liệu vẫn có thể truy cập trực tiếp như dữ liệu tự do bởi các hàm bạn, lớp bạn (friend) trong C++.

2.4.2 ASP.NET và C#.NET

Các ngôn ngữ lập trình .NET (còn được gọi là .NET Frameworks) của MicroSoft ra đời vào cuối những năm 1990 để cạnh tranh với ngôn ngữ lập trình Java. .NET là một ngôn ngữ hoàn toàn hướng đối tượng, hơn nữa, nó còn cung cấp một giao diện lập trình đồ họa thân thiện và đẹp mắt với truyền thống lập trình kéo thả của MicroSoft.

Một số đặc điểm của ngôn ngữ .NET:

Là một ngôn ngữ hoàn toàn hướng đối tượng: Tất cả các thành phần, các thực thể trong chương trình đều được mô hình dưới dạng một lớp nhất định. Không có dữ liệu tự do và hàm tự do trong chương trình.

Cung cấp giao diện lập trình đồ họa: lập trình viên chỉ cần kéo và thả các đối tượng đồ họa cho ứng dụng của mình.

Cho phép lập trình viên tự tạo ra các thư viện UserControl của mình. Đây là một thư viện bao gồm các thành phần được người dùng tự thiết kế giao diện, viết mã nguồn, đóng gói và có thể sử dụng lại trong nhiều ứng dụng khác nhau, tùy theo chức năng của các thành phần.

2.4.3 Java

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Java được xây dựng trên nền tảng của C và C++: Java sử dụng cú pháp của C và đặc trưng hướng đối tượng của C++.

Một số đặc điểm của Java:

Java là một ngôn ngữ lập trình hoàn toàn hướng đối tượng: Tất cả các thực thể đều được coi là một đối tượng, là một thể hiện cụ thể của một lớp xác định. Không có dữ liệu tự do và hàm tự do trong Java, tất cả đều được đóng gói vào các lớp xác định.

Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch thành dạng bytecode; sau đó được thực thi trên từng loại máy nhờ trình thông dịch. Điều này tạo ra khả năng hoạt động độc lập với nền tảng phần cứng của các ứng dụng Java.

Java cho phép người dùng tự tạo các đối tượng thư viện JavaBeans của mình (tương tự như các thành phần UserControl của .NET). Các đối tượng Bean sẽ được sử dụng lại như các thành phần có sẵn trong các ứng dụng khác. Điều này mở ra khả năng to lớn để tiết kiệm công sức viết mã nguồn và khả năng xây dựng các kỹ thuật cho một nền công nghiệp lắp ráp phần mềm.

Ngôn ngữ lập trình hướng đối tượng Java sẽ được trình bày chi tiết trong toàn bộ phần 2 của giáo trình này.

TỔNG KẾT CHƯƠNG 2

Nội dung chương 2 đã trình bày một số khái niệm cơ bản của lập trình hướng đối tượng:

Khái niệm đối tượng, dùng để chỉ các thực thể tồn tại thực tế trong các ứng dụng hướng đối tượng.

Khái niệm lớp, một sự trừu tượng hoá của đối tượng, dùng để biểu diễn đối tượng trong lập trình hướng đối tượng. Các thành phần của lớp là thuộc tính (dữ liệu) và phương thức (hành động).

Quá trình trừu tượng hoá theo chức năng để hình thành các phương thức của lớp, thể hiện các hoạt động của đối tượng.

Quá trình trừu tượng hoá theo dữ liệu để hình thành các thuộc tính của lớp, biểu diễn các thuộc tính tương ứng của đối tượng.

Khái niệm kế thừa trong lập trình hướng đối tượng, nhằm hạn chế việc trùng lặp mã nguồn và tăng khả năng sử dụng lại mã nguồn của chương trình.

Khái niệm đóng gói trong lập trình hướng đối tượng, nhằm hạn chế tối đa sự thay đổi mã nguồn. Chỉ cần thay đổi trong phương thức, các chương trình bên ngoài có sử dụng phương thức đó không cần phải thay đổi.

Khái niệm đa hình, cho phép gọi cùng một phương thức, nhưng với các đối tượng khác nhau sẽ có hiệu quả khác nhau.

So sánh Lớp và Cấu trúc trên các khía cạnh khác nhau: khái niệm, mục đích, chức năng và ưu nhược điểm.

Mô tả các thành phần nằm trong các vùng khác nhau của Lớp: private và public.

Giới thiệu một số ngôn ngữ lập trình hướng đối tượng thông dụng hiện nay: C++, .NET, Java.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

Trong số các nhận định sau, cái nào đúng, cái nào sai:

Đối tượng là một thực thể cụ thể, tồn tại thực tế trong các ứng dụng.

Đối tượng là một thể hiện cụ thể của Lớp.

Lớp là một khái niệm trừu tượng dùng để biểu diễn các Đối tượng.

Lớp là một sự trừu tượng hoá của Đối tượng.

Lớp và Đối tượng có bản chất giống nhau.

Trừu tượng hoá đối tượng theo chức năng tạo ra các thuộc tính của lớp.

Trừu tượng hoá đối tượng theo chức năng tạo ra các phương thức của lớp.

Trừu tượng hoá đối tượng theo dữ liệu tạo ra các thuộc tính của lớp.

Trừu tượng hoá đối tượng theo dữ liệu tạo ra các phương thức của lớp.

Kế thừa cho phép hạn chế sự trùng lặp mã nguồn.

Kế thừa cho phép tăng khả năng sử dụng lại mã nguồn.

Đóng gói hạn chế khả năng truy nhập dữ liệu.

Đóng gói hạn chế việc phải sửa đổi mã nguồn.

Đa hình cho phép thực hiện cùng một thao tác trên nhiều đối tượng khác nhau.

Liệt kê tất cả các thuộc tính và hành động của đối tượng Xe ô tô. Đề xuất lớp Car (Ô tô).

Liệt kê tất cả các thuộc tính và hành động của đối tượng Xe buýt. Đề xuất lớp Bus.

Từ hai lớp Car và Bus của bài 2 và bài 3. Đề xuất một lớp Động cơ (Engine) cho hai lớp trên kế thừa, để tránh trùng lặp dữ liệu giữa hai lớp Car và Bus.

PHẦN 2

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

CHƯƠNG 3

GIỚI THIỆU VỀ JAVA

Nội dung của chương này tập trung trình bày các vấn đề chính về ngôn ngữ lập trình Java:

- Lịch sử ra đời và phát triển của Java

- Kiến trúc tổng quát một chương trình xây dựng trên Java

- Các toán tử và các cấu trúc dữ liệu cơ bản trên Java

- Các cấu trúc lệnh của Java

3.1 LỊCH SỬ PHÁT TRIỂN CỦA JAVA

3.1.1 Java

Năm 1991, một nhóm kỹ sư của Sun Microsystems muốn lập trình để điều khiển các thiết bị điện tử như tivi, máy giặt, lò nướng... Ban đầu, họ định dùng C và C++ nhưng trình biên dịch C/C++ lại phụ thuộc vào từng loại CPU. Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, gọn, hiệu quả, độc lập thiết bị và ngôn ngữ “Oak” ra đời và vào năm 1995, sau đó được đổi tên thành Java.

Ngôn ngữ lập trình Java được Sun Microsystems đưa ra giới thiệu vào tháng 6 năm 1995 và đã nhanh chóng trở thành một ngôn ngữ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng dựa trên nền tảng của C và C++ nghĩa là Java sử dụng cú pháp của C và đặc trưng hướng đối tượng của C++. Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch thành dạng bytecode. Sau đó được thực thi trên từng loại máy nhờ trình thông dịch. Mục tiêu của các nhà thiết kế Java là cho phép người lập trình viết chương trình một lần nhưng có thể chạy trên các nền phần cứng khác nhau.

Ngày nay, Java được sử dụng rộng rãi, không chỉ để viết các ứng dụng trên máy cục bộ hay trên mạng mà còn để xây dựng các trình điều khiển thiết bị di động, PDA, ...

3.1.2 Đặc trưng của ngôn ngữ Java

Ngôn ngữ Java có những đặc trưng cơ bản sau:

- Đơn giản

- Hướng đối tượng

- Độc lập phần cứng và hệ điều hành

- Mạnh mẽ

- Bảo mật

- Phân tán

- Đa luồng

- Linh động

Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Do vậy Java loại bỏ các đặc trưng phức tạp của C và C++ như:

Loại bỏ thao tác con trỏ, thao tác định nghĩa chồng toán tử (operator overloading)...

Không cho phép đa kế thừa (Multi-inheritance) mà sử dụng các giao diện (interface)

Không sử dụng lệnh “goto” cũng như file header (.h).

Loại bỏ cấu trúc “struct” và “union”.

Hướng đối tượng

Java là ngôn ngữ lập trình hoàn toàn hướng đối tượng:

Mọi thực thể trong hệ thống đều được coi là một đối tượng, tức là một thể hiện cụ thể của một lớp xác định.

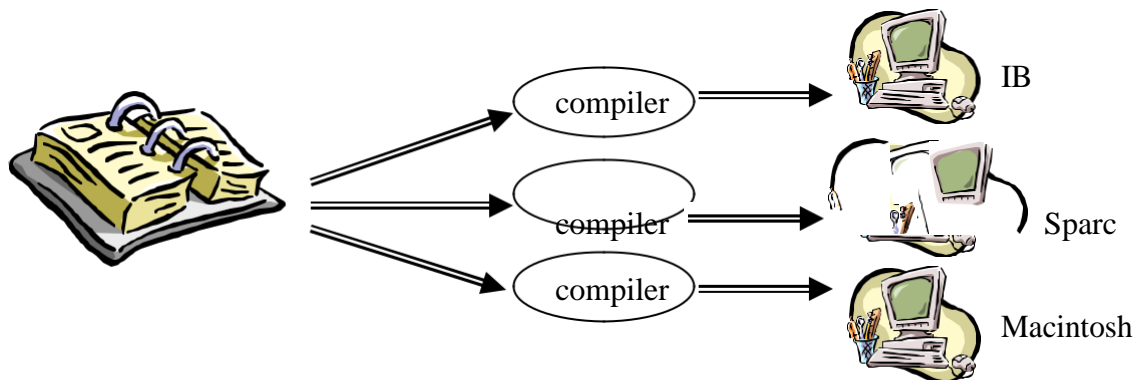
Tất cả các chương trình đều phải nằm trong một class nhất định.

Không thể dùng Java để viết một chức năng mà không thuộc vào bất kì một lớp nào.

Tức là Java không cho phép định nghĩa dữ liệu và hàm tự do trong chương trình.

Độc lập phần cứng và hệ điều hành

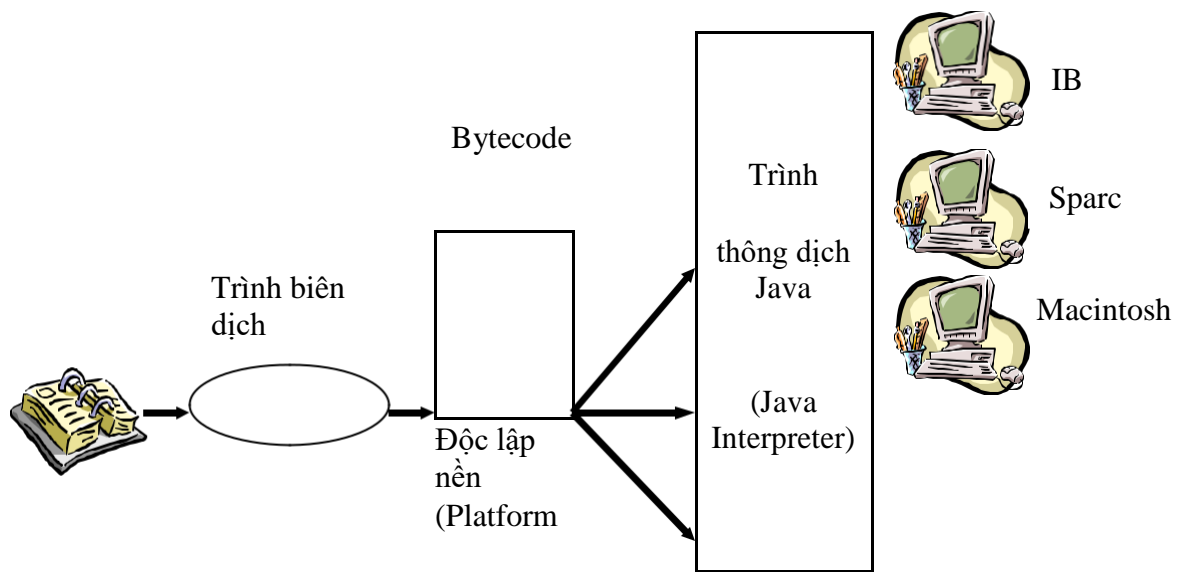
Đối với các ngôn ngữ lập trình truyền thống như C/C++, phương pháp biên dịch được thực hiện như sau (Hình 3.1): Với mỗi một nền phần cứng khác nhau, có một trình biên dịch khác nhau để biên dịch mã nguồn chương trình cho phù hợp với nền phần cứng ấy. Do vậy, khi chạy trên một nền phần cứng khác, bắt buộc phải biên dịch lại mã nguồn.



Hình 3.1 Cách biên dịch truyền thống

Đối các chương trình viết bằng Java, trình biên dịch Javac sẽ biên dịch mã nguồn thành dạng bytecode. Sau đó, khi chạy chương trình trên các nền phần cứng khác nhau, máy ảo Java dùng trình thông dịch Java để chuyển mã bytecode thành dạng chạy được trên các

nền phần cứng tương ứng. Do vậy, khi thay đổi nền phần cứng, không phải biên dịch lại mã nguồn Java. Hình 3.2 minh họa quá trình biên dịch và thông dịch mã nguồn Java.



Hình 3.2 Dịch chương trình Java

Mạnh mẽ

Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu:

Kiểu dữ liệu phải được khai báo tường minh.

Java không sử dụng con trỏ và các phép toán con trỏ.

Java kiểm tra việc truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước mảng.

Quá trình cấp phát, giải phóng bộ nhớ cho biến được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).

Cơ chế bắt lỗi của Java giúp đơn giản hóa quá trình xử lý lỗi và hồi phục sau lỗi.

Bảo mật

Java cung cấp một môi trường quản lý thực thi chương trình với nhiều mức để kiểm soát tính an toàn:

Ở mức thứ nhất, dữ liệu và các phương thức được đóng gói bên trong lớp. Chúng chỉ được truy xuất thông qua các giao diện mà lớp cung cấp.

Ở mức thứ hai, trình biên dịch kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java.

Mức thứ ba được đảm bảo bởi trình thông dịch. Chúng kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi.

Mức thứ tư kiểm soát việc nạp các lớp vào bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

Phân tán

Java được thiết kế để hỗ trợ các ứng dụng chạy trên mạng bằng các lớp Mạng (java.net). Hơn nữa, Java hỗ trợ nhiều nền chạy khác nhau nên chúng được sử dụng rộng rãi như là công cụ phát triển trên Internet, nơi sử dụng nhiều nền khác nhau.

Đa luồng

Chương trình Java cung cấp giải pháp đa luồng (Multithreading) để thực thi các công việc cùng đồng thời và đồng bộ giữa các luồng.

Linh động

Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java chứa rất nhiều thông tin thực thi nhằm kiểm soát và truy nhập đối tượng lúc chạy. Điều này cho phép khả năng liên kết động mã.

3.1.3 Cài đặt Java

Quá trình cài đặt môi trường Java trên máy bao gồm ba bước:

Copy bộ cài đặt

Chạy chương trình cài đặt

Cập nhật biến môi trường

Copy bộ cài đặt

Có thể copy từ đĩa CD hoặc tải xuống miễn phí tại địa chỉ web site của nhóm Java:

<http://www.java.sun.com/downloads/>

Chạy chương trình cài đặt

Chạy tập tin vừa copy (hoặc vừa tải về). Chọn thư mục cài đặt, thư mục mặc định là:

C:\jdk1.5 (với phiên bản JDK1.5)

Cập nhật biến môi trường

Cập nhật biến môi trường (PATH) giúp việc thực thi và biên dịch mã Java có thể tiến hành ở bất cứ một thư mục nào. Để cập nhật biến PATH, cần thêm đường dẫn đầy đủ của thư mục java vừa cài đặt (C:\jdk1.5\bin) vào cuối của giá trị biến này.

Đối với WindowsNT, WinXP khởi động Control Panel, chọn System, chọn

Environment (hoặc click chuột phải vào My Computer, chọn Properties, chọn Advanced, click vào Environment Variables), click vào biến PATH trong phần User Variables và System Variables. Sau đó, thêm vào cuối nội dung biến hiện có dòng sau (phải có dấu chấm phẩy):

;C:\jdk1.5\bin

Đối với Windows98/95, chọn START, chọn RUN, nhập dòng sysedit vào ô lệnh, nhấn

OK, chọn cửa sổ của AUTOEXEC.BAT. Tìm dòng khai báo biến PATH, nếu không có, thêm vào một dòng mới theo mẫu: SET PATH=C:\jdk1.5\bin. Nếu có sẵn

biến PATH, thêm vào cuối dòng này nội dung: ;C:\jdk1.5\bin

3.2 KIẾN TRÚC CHƯƠNG TRÌNH XÂY DỰNG TRÊN JAVA

3.2.1 Kiến trúc chương trình Java

Dạng cơ bản của một tập tin mã nguồn Java có cấu trúc như sau :

```
package packageName; // Khai báo tên gói, nếu có
import java.awt.*; // Khai báo tên thư viện sẵn có, nếu cần dùng

class className      // Khai báo tên lớp
{
    /* Đây là dòng ghi chú */
    int var;           // Khai báo biến

    public void methodName() // Khai báo tên phương
    thức {
        /* Phần thân của phương thức */
        statement (s); // Lệnh thực hiện
    }
}
```

Một tập tin mã nguồn Java có thể có ba phần chính:

Phần khai báo tên gói (khối) bằng từ khoá **package**.

Phần khai báo thư viện tham khảo bằng từ khoá **import**.

Phần khai báo nội dung lớp bằng từ khoá **class**.

Khai báo Package

Package được dùng để đóng gói các lớp trong chương trình lại với nhau thành một khối. Đây là một cách hữu hiệu để lưu trữ các lớp gần giống nhau hoặc có cùng một module thành một khối thống nhất.

Cú pháp khai báo tên gói bằng từ khoá **package**:

package <Tên gói>;

Để đặt tên package trong chương trình, người ta có thể tiến hành như đặt tên thư mục trên ổ đĩa. Nghĩa là bắt đầu bằng tên có phạm vi lớn, cho đến các tên có phạm vi nhỏ, cuối

cùng là tên các gói trực tiếp chứa các lớp. Phạm vi đặt tên gói, trên thực tế, được tiến hành theo thứ tự phạm vi lớn đến nhỏ như sau:

Tên tổ chức

Tên công ty

Tên dự án

Tên modul trong dự án

Tên các chức năng trong modul.

Ví dụ:

Tên miền của công ty là **syz.com**

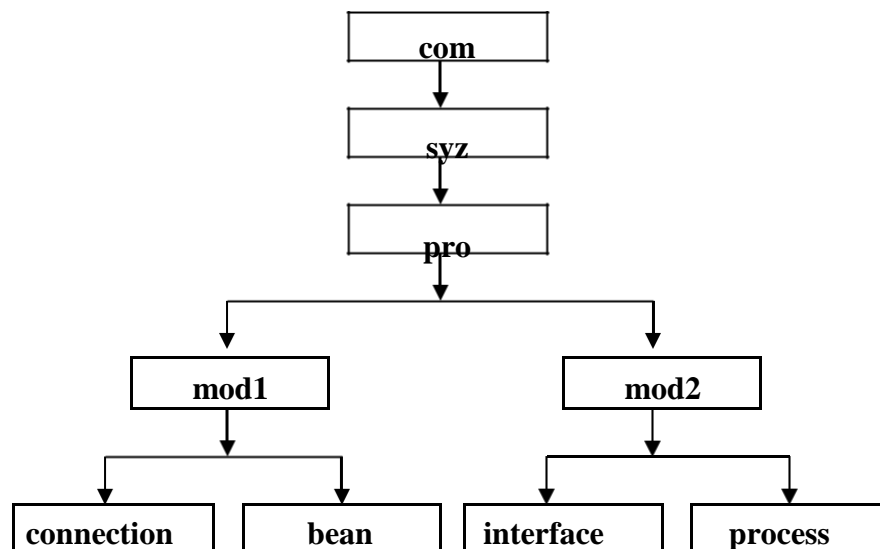
Tên dự án là **pro**

Dự án có hai modul là **mod1** và **mod2**

Modul mod1 có hai chức năng là kết nối cơ sở dữ liệu **connection** và biểu diễn dữ liệu **bean**.

Modul mod2 có hai chức năng là giao tiếp **interface** và xử lý yêu cầu **process**.

Khi đó, cấu trúc khối của dự án được mô tả như hình 3.3



Hình 3.3: Kiến trúc khối của dự án

Khi đó, trong chức năng **bean** có lớp User, thì phải khai báo tên khối trong lớp này như sau:

```
package com.syz.pro.mod1.bean;
```

Ưu điểm của package:

Cho phép nhóm các lớp vào với nhau thành các đơn vị nhỏ hơn. Việc thao tác trên các đơn vị khối sẽ gọn hơn thao tác trên một tập các lớp.

Tránh việc xung đột khi đặt tên lớp. Vì các lớp không cùng package thì có thể đặt tên trùng nhau. Khi số lượng lớp của chương trình quá lớn ta có thể tránh phải đặt tên khác nhau cho các lớp bằng cách đặt chúng vào các package khác nhau.

Cho phép bảo vệ các lớp. Khi chương trình lớn, việc chia nhỏ chương trình thành các package sẽ thuận lợi hơn cho việc quản lí và phát triển.

Tên gói còn được dùng để định danh lớp trong ứng dụng.

Lưu ý:

Dòng lệnh khai báo tên khối phải được đặt đầu tiên trong tên tin mã chương trình.

Chỉ được khai báo tối đa một tên khối đối với mỗi tệp mã nguồn Java.

Các tệp tin của các lớp nằm cùng gói ứng dụng phải được lưu trong cùng một thư mục (tên thư mục là tên khối) theo cấu trúc khối của dự án.

Tên khối nên đặt theo chữ thường vì tên khối sẽ là tên thư mục tương ứng trong ổ đĩa, tránh nhầm lẫn với tên các tệp tin là tên các lớp của chương trình.

Khi không phân chia chương trình thành khối (chương trình đơn giản), không cần thiết phải khai báo tên khối ở đầu chương trình.

Khai báo thư viện

Khai báo thư viện để chỉ ra những thư viện đã được định nghĩa sẵn mà chương trình sẽ tham khảo tới. Cú pháp khai báo thư viện với từ khoá **import** như sau:

```
import <Tên thư viện>;
```

Java chuẩn cung cấp một số thư viện như sau:

java.lang: cung cấp các hàm thao tác trên các kiểu dữ liệu cơ bản, xử lí lỗi và ngoại lệ, xử lí vào ra trên các thiết bị chuẩn như bàn phím và màn hình.

java.applet: cung cấp các hàm cho xây dựng các applet (sẽ trình bày trong Chương 6).

java.awt: cung cấp các hàm cho xây dựng các ứng dụng đồ hoạ với các thành phần giao diện multi media (sẽ trình bày chi tiết trong Chương 6).

java.io: cung cấp các hàm xử lí vào/ra trên các thiết bị chuẩn và các thiết bị ngoại vi.

java.util: cung cấp các hàm tiện ích trong xử lí liên quan đến các kiểu dữ liệu có cấu trúc như Date, Stack, Vector.

Ví dụ, nếu trong chương trình cần đến các thao tác chuyển kiểu đổi dữ liệu tương minh (từ kiểu string sang kiểu int), thì ta sẽ phải tham khảo thư viện **java.lang**:

```
import java.lang.*;
```

Lưu ý:

Nếu muốn khai báo tham khảo nhiều thư viện, phải khai báo tham khảo mỗi thư viện với một từ khoá **import**.

Nếu chỉ tham khảo một vài lớp trong một thư viện, nên chỉ rõ tham khảo lớp nào, thay vì phải khai báo tham khảo cả gói (bằng kí hiệu “*”) vì tham khảo cả gói sẽ tăng kích cỡ tệp tin class sau khi biên dịch.

Nếu không tham khảo thư viện nào, không cần thiết phải khai báo các tham khảo với từ khoá **import**.

Khai báo lớp

Phần thứ ba là phần khai báo lớp và nội dung của lớp, phần này luôn bắt buộc phải có đối với một tệp mã nguồn Java:

Khai báo tên lớp với từ khoá **class**.

Khái báo các thuộc tính của lớp.

Khai báo các phương thức của lớp

Việc khai báo lớp với các thuộc tính và phương thức sẽ được trình bày chi tiết trong chương 4.

3.2.2 Chương trình Java đầu tiên

Chương trình sau đây cho phép hiển thị một thông điệp (Nằm trong tệp mã nguồn First.java):

Chương trình 3.1

```
package vidu.chuong3;
// Đây là chương trình “First.java”
class First
{
    public static void main(String args[])
    {
        System.out.println(“Hello World”);
    }
}
```

Để biên dịch mã nguồn, ta sử dụng trình biên dịch **javac**. Trình biên dịch xác định tên của file nguồn tại dòng lệnh như mô tả dưới đây (giả sử đang ở thư mục chứa package vidu và biến môi trường PATH đã được thiết lập đúng qui cách):

>javac vidu/chuong3/First.java

Trình dịch javac tạo ra file First.class chứa các mã “bytecodes”. Những mã này chưa thể thực thi được. Để chương trình thực thi được ta cần dùng trình thông dịch “**java interpreter**” với lệnh **java**. Lệnh được thực hiện như sau:

```
>javac vidu.chuong3.First
```

Kết quả sẽ hiển thị trên màn hình như sau:

Hello World

3.2.3 Phân tích chương trình đầu tiên

Trong Java, tất cả mã lệnh đều phải được tổ chức vào trong một lớp nhất định. Do đó, mỗi tệp tin mã nguồn xác định ít nhất một lớp Java và tên tệp tin phải trùng với tên lớp. Java phân biệt chữ hoa và chữ thường, cho nên tên tệp tin của chương trình trên phải trùng với tên lớp: First.java.

```
package vidu.chuong3;
```

Đây là dòng khai báo tên khối của chương trình, vì tên khối của chương trình được đặt theo hai mức:

Mức thứ nhất là kiểu bài: ví dụ (vidu) hoặc bài tập (baitap).

Mức thứ hai là tên của chương: chuong3, chuong4, chuong5, chuong6.

Vì đây là ví dụ, nằm ở chương 3 nên thuộc vào gói **vidu.chuong3**. Đồng thời, tệp tin First.java sẽ nằm trong thư mục: ../vidu/chuong3/.

Chương trình này không tham khảo thư viện nào nên không cần lệnh import nào.

```
// Đây là chương trình “First.java”
```

Ký hiệu “//” dùng để chú thích dòng lệnh. Trình biên dịch sẽ bỏ qua dòng chú thích này.

Java hỗ trợ hai loại chú thích:

Loại chú thích trên một dòng, dùng “//”. Trình biên dịch sẽ bỏ qua nội dung bắt đầu từ ký hiệu “//” cho đến hết dòng lệnh chứa nó.

Loại chú thích trên nhiều dòng có thể bắt đầu với “/*” và kết thúc với “*/”. Trình biên dịch sẽ bỏ qua nội dung nằm giữa hai ký hiệu này.

Dòng kế tiếp khai báo lớp có tên **First**: Bắt đầu với từ khoá **class**, kế đến là tên lớp

```
class First
```

Một định nghĩa lớp nằm trọn vẹn giữa hai ngoặc móc mở “{” và đóng “}”. Các ngoặc này đánh dấu bắt đầu và kết thúc một khối lệnh.

```
public static void main(String args[ ])
```

Đây là phương thức chính, từ đây chương trình bắt đầu việc thực thi của mình. Tất cả các ứng dụng java đều sử dụng một phương thức **main** này.

Từ khoá **public** là một chỉ định truy xuất. Nó cho biết thành viên của lớp có thể được truy xuất từ bất cứ đâu trong chương trình.

Từ khoá **static** cho phép **main** được gọi tới mà không cần tạo ra một thể hiện (instance) của lớp. Nó không phụ thuộc vào các thể hiện của lớp được tạo ra.

Từ khoá **void** thông báo cho máy tính biết rằng phương thức sẽ không trả lại bất cứ giá trị nào khi thực thi chương trình.

String args[] là tham số dùng trong phương thức **main**. Khi không có một thông tin nào được chuyển vào **main**, phương thức được thực hiện với các dữ liệu rỗng – không có gì trong dấu ngoặc đơn.

System.out.println(“Hello World”); Dòng lệnh này hiển thị chuỗi **“Hello World”** trên màn hình. Lệnh **println()** cho phép hiển thị chuỗi được truyền vào lên màn hình.

Truyền đối số trong dòng lệnh

Chương trình 3.2 minh hoạ các tham số (argument) của các dòng lệnh được tiếp nhận như thế nào trong phương thức **main**.

Chương trình 3.2

```
package vidu.chuong3;
class PassArgument{
    public static void main(String args[])
    {
        System.out.println(“This is what the main method received”);
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
    }
}
```

Biên dịch chương trình:

>javac PassArgument.java

Thực thi chương trình với dòng lệnh:

>java PassArgument A 123 B1

Sẽ thu được trên màn hình kết quả:

```
This is what the main method received
A
123
B1
```

3.3 CÁC KIỂU DỮ LIỆU VÀ TOÁN TỬ CƠ BẢN TRÊN JAVA

3.3.1 Khai báo biến

Cú pháp khai báo biến:

dataType varName;

Trong đó, dataType là kiểu dữ liệu của biến, varName là tên biến. Trong Java, việc đặt tên biến phải tuân theo các quy tắc sau:

Chỉ được bắt đầu bằng một kí tự (chữ), hoặc một dấu gạch dưới , hoặc một kí tự dollar

Không có khoảng trắng giữa tên

Bắt đầu từ kí tự thứ hai, có thể dùng các kí tự (chữ), chữ số, dấu dollar, dấu gạch dưới

Không trùng với các từ khoá

Có phân biệt chữ hoa chữ thường

Phạm vi hoạt động của biến

Một biến có phạm vi hoạt động trong toàn bộ khối lệnh mà nó được khai báo. Một khối lệnh bắt đầu bằng dấu “{” và kết thúc bằng dấu “}”:

Nếu biến được khai báo trong một cấu trúc lệnh điều khiển, biến đó có phạm vi hoạt động trong khối lệnh tương ứng.

Nếu biến được khai báo trong một phương thức (Không nằm trong khối lệnh nào), biến đó có phạm vi hoạt động trong phương thức tương ứng: có thể được sử dụng trong tất cả các khối lệnh của phương thức.

Nếu biến được khai báo trong một lớp (Không nằm trong một phương thức nào), biến đó có phạm vi hoạt động trong toàn bộ lớp tương ứng: có thể được sử dụng trong tất cả các phương thức của lớp.

3.3.2 Kiểu dữ liệu

Trong Java, kiểu dữ liệu được chia thành hai loại:

Các kiểu dữ liệu cơ bản

Các kiểu dữ liệu đối tượng

Kiểu dữ liệu cơ bản

Java cung cấp các kiểu dữ liệu cơ bản như sau:

byte: Dùng để lưu dữ liệu kiểu số nguyên có kích thước một byte (8 bit). Phạm vi biểu diễn giá trị từ -128 đến 127. Giá trị mặc định là 0.

char: Dùng để lưu dữ liệu kiểu kí tự hoặc số nguyên không âm có kích thước 2 byte (16 bit). Phạm vi biểu diễn giá trị từ 0 đến u\ffff. Giá trị mặc định là 0.

boolean: Dùng để lưu dữ liệu chỉ có hai trạng thái đúng hoặc sai (độ lớn chỉ có 1 bit). Phạm vi biểu diễn giá trị là {“True”, “False”}. Giá trị mặc định là False.

short: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 2 byte (16 bit). Phạm vi biểu diễn giá trị từ - 32768 đến 32767. Giá trị mặc định là 0.

int: Dùng để lưu dữ liệu có kiểu số nguyên, kích cỡ 4 byte (32 bit). Phạm vi biểu diễn giá trị từ -2,147,483,648 đến 2,147,483,647. Giá trị mặc định là 0.

- float:** Dùng để lưu dữ liệu có kiểu số thực, kích cỡ 4 byte (32 bit). Giá trị mặc định là 0.0f.
- double:** Dùng để lưu dữ liệu có kiểu số thực có kích thước lên đến 8 byte. Giá trị mặc định là 0.00d
- long:** Dùng để lưu dữ liệu có kiểu số nguyên có kích thước lên đến 8 byte. Giá trị mặc định là 0l.

Kiểu dữ liệu đối tượng

Trong Java, có 3 kiểu dữ liệu đối tượng:

- Array:** Một mảng của các dữ liệu cùng kiểu
- class:** Dữ liệu kiểu lớp đối tượng do người dùng định nghĩa. Chứa tập các thuộc tính và phương thức.
- interface:** Dữ liệu kiểu lớp giao tiếp do người dùng định nghĩa. Chứa các phương thức của giao tiếp.

Ép kiểu (Type casting)

Ví dụ, nhiều khi gặp tình huống cần cộng một biến có dạng **integer** với một biến có dạng **float**. Để xử lý tình huống này, Java sử dụng tính năng ép kiểu (type casting) của C/C++. Đoạn mã sau đây thực hiện phép cộng một giá trị dấu phẩy động (float) với một giá trị nguyên (integer).

```
float c = 35.8f;  
int b = (int)c + 1;
```

Đầu tiên giá trị dấu phẩy động **c** được đổi thành giá trị nguyên 35. Sau đó nó được cộng với 1 và kết quả là giá trị 36 được lưu vào **b**.

Trong Java có hai loại ép kiểu dữ liệu:

Nới rộng (widening): quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin. Ví dụ chuyển từ **int** sang **float**. Chuyển kiểu loại này có thể được thực hiện ngầm định bởi trình biên dịch.

Thu hẹp (narrowwing): quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin như ví dụ ở trên. Chuyển kiểu loại này không thể thực hiện ngầm định bởi trình biên dịch, người dùng phải thực hiện chuyển kiểu tường minh.

3.3.3 Các toán tử

Java cung cấp các dạng toán tử sau:

- Toán tử số học
- Toán tử bit
- Toán tử quan hệ
- Toán tử logic
- Toán tử điều kiện
- Toán tử gán

Toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu boolean không sử dụng được, song các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Toán tử	Mô tả
+	Cộng. Trả về giá trị tổng hai toán hạng
-	Trừ Trả về kết quả của phép trừ.
*	Nhân Trả về giá trị là tích hai toán hạng.
/	Chia Trả về giá trị là thương của phép chia
%	Phép lấy modul Giá trị trả về là phần dư của phép chia
++	Tăng dần Tăng giá trị của biến lên 1. Ví dụ <code>a++</code> tương đương với <code>a = a + 1</code>
--	Giảm dần Giảm giá trị của biến 1 đơn vị. Ví dụ <code>a--</code> tương đương với <code>a = a - 1</code>
+=	Cộng và gán giá trị Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c += a</code> tương đương <code>c = c + a</code>
-=	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c -= a</code> tương đương với <code>c = c - a</code>
*=	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c *= a</code> tương đương với <code>c = c*a</code>
/=	Chia và gán

	Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c /= a$ tương đương với $c = c/a$
$\% =$	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị số dư vào toán hạng bên trái. Ví dụ $c \% = a$ tương đương với $c = c \% a$

Bảng 3.1 Các toán tử số học

Toán tử Bit

Các toán tử dạng bit cho phép ta thao tác trên từng bit riêng biệt trong các kiểu dữ liệu nguyên thủy.

Toán tử	Mô tả
\sim	Phủ định bit (NOT) Trả về giá trị phủ định của một bit.
$\&$	Toán tử AND bit Trả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác
$ $	Toán tử OR bit Trả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác.
\wedge	Toán tử Exclusive OR bit Trả về giá trị là 1 nếu chỉ một trong các toán hạng là 1 và trả về 0 trong các trường hợp khác.
$>>$	Dịch sang phải bit

	Chuyển toàn bộ các bit của một số sang phải một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch.
	Dịch sang trái bit Chuyển toàn bộ các bit của một số sang trái một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch.

Bảng 3.2 Các toán tử Bit

Các toán tử quan hệ

Các toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic “đúng” hoặc “sai”). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Toán tử	Mô tả
=	So sánh bằng Toán tử này kiểm tra sự tương đương của hai toán hạng
!=	So sánh khác Kiểm tra sự khác nhau của hai toán hạng
>	Lớn hơn Kiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay không
<	Nhỏ hơn Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không

Bảng 3.3 Các toán tử quan hệ

Các toán tử logic: Các toán tử logic làm việc với các toán hạng Boolean. Một vài toán tử kiểu này được chỉ ra dưới đây

Toán tử	Mô tả
&&	Và (AND) Trả về một giá trị “Đúng” (True) nếu chỉ khi cả hai toán tử có giá trị “True”
	Hoặc (OR) Trả về giá trị “True” nếu ít nhất một giá trị là True
^	XOR Trả về giá trị True nếu và chỉ nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai)
!	Toán hạng đơn tử NOT. Chuyển giá trị từ True sang False và ngược lại.

Bảng 3.4 Các toán tử logic

Các toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó bao gồm ba thành phần cấu thành biểu thức điều kiện. Cú pháp:

<biểu thức 1> ? <biểu thức 2> : <biểu thức 3>;

biểu thức 1: Biểu thức logic. Trả trả về giá trị True hoặc False

biểu thức 2: Là giá trị trả về nếu <biểu thức 1> xác định là True

biểu thức 3: Là giá trị trả về nếu <biểu thức 1> xác định là False

Toán tử gán

Toán tử gán (=) dùng để gán một giá trị vào một biến và có thể gán nhiều giá trị cho nhiều biến cùng một lúc. Ví dụ đoạn lệnh sau gán một giá trị cho biến **var** và giá trị này lại được gán cho nhiều biến trên một dòng lệnh đơn.

```
int var = 20;
int p,q,r,s;
p=q=r=s=var;
```

Dòng lệnh cuối cùng được thực hiện từ phải qua trái. Đầu tiên giá trị ở biến var được gán cho ‘s’, sau đó giá trị của ‘s’ được gán cho ‘r’ và cứ tiếp như vậy.

Thứ tự ưu tiên của các toán tử

Các biểu thức được viết ra nói chung gồm nhiều toán tử. Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java

Thứ tự	Toán tử
--------	---------

1.	Các toán tử đơn như +,-,++,--
2.	Các toán tử số học và các toán tử dịch như *,/,+,-,<<,>>
3.	Các toán tử quan hệ như >,<,>=,<=,==,!=
4.	Các toán tử logic và Bit như &&,,&,, ^
5.	Các toán tử gán như =,*=,/=,+=,-=

Bảng 3.5 Thứ tự ưu tiên các toán tử

Thay đổi thứ tự ưu tiên

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn ():

Phần được giới hạn trong ngoặc đơn được thực hiện trước.

Nếu dùng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài.

Trong phạm vi một cặp ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

3.4 CÁC CẤU TRÚC LỆNH TRÊN JAVA

Java cung cấp hai loại cấu trúc điều khiển:

Điều khiển rẽ nhánh

Mệnh đề if-else

Mệnh đề swich-case

Vòng lặp (Loops)

Vòng lặp while

Vòng lặp do-while

Vòng lặp for

3.4.1 Câu lệnh if-else

kiểm tra giá trị dạng boolean của điều kiện. Nếu giá trị điều kiện là **True** Câu lệnh **if-else** lệnh sau **if** sẽ được thực hiện, nếu là **False** thì chỉ có khối lệnh sau **else** Chỉ có khối pháp: được thực hiện.

```

if (conditon)
{
    action1 statements;
}
else
{
    action2 statements;
}

```

```
}
```

Condition: Biểu thức boolean như toán tử so sánh.

action 1: Khối lệnh được thực thi khi giá trị điều kiện là True

action 2: Khối lệnh được thực thi nếu điều kiện trả về giá trị False

Đoạn chương trình sau kiểm tra xem các số có chia hết cho 5 hay không.

Chương trình 3.3

```
package vidu.chuong3;
class CheckNumber
{
    public static void main(String args[])
    {
        int num = 10;
        if(num%5 == 0)
            System.out.println (num + " is divisible for 5!");
        else
            System.out.println (num + " is indivisible for 5!");
    }
}
```

đoạn chương trình trên num được gán giá trị nguyên là 10. Trong câu lệnh **if-else** điều kiện **num %5** trả về giá trị 0 và điều kiện thực hiện là True. Thông báo “10 is divisible for 5!” được in ra. Lưu ý rằng vì chỉ có một câu lệnh được viết trong đoạn “if” và “else”, bởi vậy không cần thiết phải được đưa vào dấu ngoặc móc “{” và “}”.

3.4.2 Câu lệnh switch-case

Khối lệnh switch-case có thể được sử dụng thay thế câu lệnh if-else trong trường hợp một biểu thức cho ra nhiều kết quả. Cú pháp:

```
swich (expression)
{
    case 'value1': action 1 statement;
                    break;
    case 'value2': action 2 statement;
                    break;
    .....
    case 'valueN': actionN statement;
                    break;
```

default: default_action statement;

}

expression - Biến chứa một giá trị xác định

value1,value 2,...valueN: Các giá trị hằng số phù hợp với giá trị trên biến **expression**

.

action1,action2...actionN: Khối lệnh được thực thi khi trường hợp tương ứng có giá trị True

break: Từ khoá được sử dụng để bỏ qua tất cả các câu lệnh sau đó và giành quyền điều khiển cho cấu trúc bên ngoài **switch**

default: Từ khóa tùy chọn được sử dụng để chỉ rõ các câu lệnh nào được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

default - action: Khối lệnh được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

Đoạn chương trình sau xác định giá trị trong một biến nguyên và hiển thị ngày trong tuần được thể hiện dưới dạng chuỗi. Để kiểm tra các giá trị nằm trong khoảng từ 0 đến 6, chương trình sẽ thông báo lỗi nếu nằm ngoài phạm vi trên.

Chương trình 3.4

```
package vidu.chuong3;
class SwitchDemo
{
    public static void main(String agrs[])
    {
        int day = 2;
        switch(day)
        {
            case 0 : System.out.println("Sunday");
                        break;
            case 1 : System.out.println("Monday");
                        break;
            case 2 : System.out.println("Tuesday");
                        break;
            case 3 : System.out.println("Wednesday");
                        break;
            case 4 : System.out.println("Thursday");
                        break;
            case 5 : System.out.println("Friday");
```

```

        break;
    case 6 : System.out.println("Satuday");
        break;
    default:
        System.out.println("Invalid day of week");
    }
}
}

```

Nếu giá trị của biến **day** là 2, chương trình sẽ hiển thị **Tuesday**, và cứ tiếp như vậy .

3.4.3 Vòng lặp While

Vòng lặp **while** thực thi khối lệnh khi điều kiện thực thi vẫn là True và dừng lại khi điều kiện thực thi nhận giá trị False. Cú pháp:

```

while(condition)
{
    action statements;
}

```

condition: có giá trị bool; vòng lặp sẽ tiếp tục cho nếu điều kiện vẫn có giá trị True.

action statement: Khối lệnh được thực hiện nếu **condition** nhận giá trị True

Đoạn chương trình sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc while.

Chương trình 3.5

```

package vidu.chuong3;
class WhileDemo
{
    public static void main(String args[])
    {
        int a = 5, sum = 1;
        while (a >= 1)
        {
            sum +=a;
            a--;
        }
        System.out.println("The sum is " + sum);
    }
}

```


ví dụ trên, vòng lặp được thực thi cho đến khi điều kiện $a \geq 1$ là **True**. Biến **a** được khai báo bên ngoài vòng lặp và được gán giá trị là 5. Cuối mỗi vòng lặp, giá trị của **a** giảm đi 1. Sau năm vòng giá trị của **a** bằng 0. Điều kiện trả về giá trị False và vòng lặp kết thúc. Kết quả sẽ được hiển thị “**The sum is 15**”

3.4.4 Vòng lặp do-while

Vòng lặp do-while thực thi khối lệnh khi mà điều kiện là True, tương tự như vòng lặp while, ngoại trừ do-while thực hiện lệnh ít nhất một lần ngay cả khi điều kiện là False. Cú pháp:

```
do{  
    action statements;  
}while(condition);
```

condition: Biểu thức bool; vòng lặp sẽ tiếp tục khi mà điều kiện vẫn có giá trị True.

action statement: Khối lệnh luôn được thực hiện ở lần thứ nhất, từ vòng lặp thứ hai, chúng được thực hiện khi **condition** nhận giá trị True.

Ví dụ sau tính tổng của 5 số tự nhiên đầu tiên dùng cấu trúc do-while.

Chương trình 3.6

```
package vidu.chuong3;  
class DoWhileDemo  
{  
    public static void main(String args[])  
    {  
        int a = 1, sum = 0;  
        do{  
            sum += a;  
            a++;  
        }while (a <= 5);  
        System.out.println("Sum of 1 to 5 is " + sum);  
    }  
}
```

Biến **a** được khởi tạo với giá trị 1, sau đó nó vừa được dùng làm biến chạy (tăng lên 1 sau mỗi lần lặp) vừa được dùng để cộng dồn vào biến **sum**. Tại thời điểm kết thúc, chương trình sẽ in ra **Sum of 1 to 5 is 15**.

3.4.5 Vòng lặp for

Vòng lặp for cung cấp một dạng kết hợp tất cả các đặc điểm chung của tất cả các loại vòng lặp: giá trị khởi tạo của biến chạy, điều kiện dừng của vòng lặp và lệnh thay đổi giá trị của biến chạy. Cú pháp:

```
for(initialization statements; condition; increment statements)
{
    action statements;
}
```

initialization statements: khởi tạo giá trị ban đầu cho các biến chạy, các lệnh khởi tạo được phân cách nhau bởi dấu phẩy và chỉ thực hiện duy nhất một lần vào thời điểm bắt đầu của vòng lặp.

condition: Biểu thức bool; vòng lặp sẽ tiếp tục cho đến khi nào điều kiện có giá trị False.

increment statements: Các câu lệnh thay đổi giá trị của biến chạy. Các lệnh này luôn được thực hiện sau mỗi lần thực hiện khối lệnh trong vòng lặp. Các lệnh phân biệt nhau bởi dấu phẩy.

Đoạn chương trình sau hiển thị tổng của 5 số đầu tiên dùng vòng lặp for.

Chương trình 3.7

```
package vidu.chuong3;
class ForDemo
{
    public static void main(String args[])
    {
        int sum = 0;
        for (int i=1; i<=5; i++)
            sum += i;
        System.out.println ("The sum is " + sum);
    }
}
```

ví dụ trên, i và sum là hai biến được gán các giá trị đầu là 1 và 0 tương ứng. Điều kiện được kiểm tra và khi nó còn nhận giá trị True, câu lệnh tác động trong vòng lặp được thực hiện. Tiếp theo giá trị của i được tăng lên 2 để tạo ra số chẵn tiếp theo. Một lần nữa, điều kiện lại được kiểm tra và câu lệnh tác động lại được thực hiện. Sau năm vòng, i tăng lên 6,

điều kiện trả về giá trị False và vòng lặp kết thúc. Thông báo: **The sum is 15** được hiển thị.

3.5 CASE STUDY I

Bây giờ, áp dụng các nội dung đã học trong chương này để viết một chương trình tính chu và diện tích của một hình chữ nhật có kích thước x, y với yêu cầu:

Kích thước x, y nhập từ tham số dòng lệnh.

Phải kiểm tra x, y là các số nguyên dương hay không trước khi tính toán. In kết quả tính toán ra màn hình

Đây là đoạn chương trình thực hiện bài toán này.

Chương trình 3.8

```
package vidu.chuong3;

import java.awt.*;
import java.lang.*;

class RectangleDemo
{
    public static void main(String args[])
    {
        //khai báo các biến lưu giữ kích thước của hình chữ
        nhật int x = 0, y = 0;

        /*đọc các kích thước từ tham số dòng lệnh*/
        //nếu truyền đủ hai tham số thì mới tính tiếp
        if(args.length >= 2)
        {
            //chuyển kiểu từ String sang integer
            x = Integer.parseInt(args[0]);
            y = Integer.parseInt(args[1]);
        }

        /*Tính chu vi và diện tích hình chữ nhật*/
        //nếu cả hai tham số đều dương thì mới tính
        if(x>0 && y>0)
        {
            //tính chu vi
            int chuvi = 2*(x + y);
            System.out.println ("Chu vi là " + chuvi);
            //tính diện tích
            int dientich = x*y;
            System.out.println ("Diện tích là " + dientich);
        }
        else
            System.out.println ("Các tham số không đúng!");
    }
}
```

Sau khi biên dịch chương trình 3.8 (tệp tin có tên RectangleDemo.java), ta chạy từ cửa sổ dòng lệnh:

>java RectangleDemo 10 20

Sẽ thu được kết quả:

Chu vi là: 60

Diện tích là: 200

Nếu chỉ gõ ở cửa sổ dòng lệnh:

>java RectangleDemo

Thì sẽ nhận được một thông báo lỗi:

Các tham số không đúng!

TỔNG KẾT CHƯƠNG 3

Nội dung chương 3 đã trình bày các nội dung cơ bản về cú pháp ngôn ngữ lập trình Java:

Tất cả các lệnh của java phải được tổ chức vào trong một lớp nhất định. Tên tập tin mã nguồn phải trùng với tên lớp.

Lệnh **package** được dùng để khai báo tên gói của lớp.

Lệnh **import** được sử dụng trong chương trình để truy cập các gói thư viện Java.

Lệnh **class** được dùng để khai báo tên lớp

Tên lớp, tên phương thức, tên hằng và tên biến trong java phải tuân theo quy tắc đặt tên của java.

Ứng dụng Java có một lớp chứa phương thức main. Các tham số có thể được truyền vào phương thức main nhờ các tham số lệnh (command line parameters).

Java cung cấp các dạng toán tử:

Các toán tử số học

Các toán tử bit

Các toán tử quan hệ

Các toán tử logic

Toán tử điều kiện

Toán tử gán

Java cung cấp các cấu trúc điều khiển lệnh:

if-else

switch

for

while

do while

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

Trong các tên sau, tên nào có thể dùng làm tên biến trong java:

_123

a\$

1abc

class

vi du

\$123

Muốn lưu giữ một biến số nguyên dương mà có giá trị lớn nhất là một triệu thì dùng kiểu dữ liệu nào là tiết kiệm bộ nhớ nhất?

Muốn lưu giữ một biến số nguyên âm mà có giá trị nhỏ nhất là âm một tỉ thì dùng kiểu dữ liệu nào là tiết kiệm bộ nhớ nhất?

Trong cấu trúc lệnh if-else đơn (1 if và 1 else) thì có ít nhất một khối lệnh (của if hoặc của else) được thực hiện. Đúng hay sai?

Trong cấu trúc lệnh switch-case, khi không dùng “default” thì có ít nhất một khối lệnh được thực hiện. Đúng hay sai?

Trong cấu trúc lệnh switch-case, khi dùng “default” thì có ít nhất một khối lệnh được thực hiện. Đúng hay sai?

Trong cấu trúc lệnh while, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?

Trong cấu trúc lệnh do-while, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?

Trong cấu trúc lệnh for, khối lệnh được thực hiện ít nhất một lần ngay cả khi điều kiện có giá trị False. Đúng hay sai?

Cho biết kết quả thu được khi thực hiện đoạn chương trình sau?

```
class me{  
    public static void main(String args[]){  
        int sales = 820;  
        int profit = 200;  
        System.out.println((sale +profit)/10*5);  
    }  
}
```

Cho biết đoạn chương trình sau thực hiện vòng lặp bao nhiêu lần và kết quả in ra là gì?

```
class me{
    public static void main(String args[]){
        int i = 0;
        int sum = 0;
        do{
            sum += i;
            i++;
        }while(i <= 10);
        System.out.println(sum);
    }
}
```

Cho biết đoạn chương trình sau thực hiện vòng lặp bao nhiêu lần và kết quả in ra là gì?

```
class me{
    public static void main(String args[]){
        int i = 5;
        int sum = 0;
        do{
            sum += i;
            i++;
        }while(i < 5);
        System.out.println(sum);
    }
}
```

Cho biết hai đoạn chương trình sau in ra kết quả giống hay khác nhau?

```
class me1{
    public static void main(String args[]){
        int i = 0;
        int sum = 0;
        for(i=0; i<5; i++){
            sum += i;
        }
        System.out.println(sum);
    }
}
```

và:

```
class me2{
    public static void main(String args[]){
        int i = 0;
        int sum = 0;
        for( ; i<5; i++){
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Viết chương trình tính tổng các số chẵn nằm trong khoảng 1 đến 100.

Viết chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.

Viết chương trình tìm giai thừa của n ($n > 0$), n nhập từ tham số dòng lệnh.

Viết chương trình tìm bội số chung nhỏ nhất của m và n ($m, n > 0$), m và n được nhập từ tham số dòng lệnh.

Viết chương trình tìm ước số chung lớn nhất của m và n ($m, n > 0$), m và n được nhập từ tham số dòng lệnh.

Viết chương trình tìm số Fibonacci thứ n ($n > 2$), n nhập từ tham số dòng lệnh. Biết rằng số Fibonacci được tính theo công thức: $F(n) = F(n-1) + F(n-2)$ với $n \geq 2$ và $F(0) = F(1) = 1$.

CHƯƠNG 4

KẾ THỪA VÀ ĐA HÌNH TRÊN JAVA

Nội dung của chương này tập trung trình bày các đặc trưng hướng đối tượng của ngôn ngữ Java:

- Kế thừa đơn
- Kế thừa kép
- Các lớp trừu tượng
- Đa hình

4.1 KẾ THỪA ĐƠN

4.1.1 Lớp

Java coi lớp là một khuôn mẫu (Template) của một đối tượng, trong đó lớp chứa các thuộc tính và các phương thức hoạt động của đối tượng.

Khai báo lớp

Một lớp được khai báo với cú pháp:

```
<tính chất> class <tên lớp>
{
}
```

Lớp trong java có ba tính chất đặc trưng bởi ba từ khoá:

public: Lớp thông thường, có thể được truy cập từ các gói (package) khác. **public** là giá trị mặc định cho tính chất của lớp.

final: Khai báo lớp hằng, lớp này không thể tạo dẫn xuất. Tức là không có lớp nào kế thừa được từ các lớp có tính chất final.

abstract: Khai báo lớp trừu tượng, lớp này chỉ được phép chứa các phương thức trừu tượng. Hơn nữa, không thể tạo các thể hiện (Instance) của các lớp trừu tượng bằng toán tử **new** như các lớp thông thường.

Chương trình 4.1 khai báo một lớp thông thường với kiểu mặc định là public với dòng khai báo.

Chương trình 4.1

```
package vidu.chuong4;
class Person
{
}
```

Sử dụng lớp

Lớp được sử dụng khi chương trình cần một đối tượng có kiểu của lớp đó. Khi đó, đối tượng được khai báo dựa vào toán tử **new**:

<tên lớp> <tên đối tượng> = new <tên lớp>();

Ví dụ, muốn tạo một đối tượng có kiểu là lớp Person trong chương trình 4.1, ta dùng lệnh sau:

Person myClass = new Person();

Khai báo thuộc tính của lớp

Thuộc tính của lớp được khai báo theo cú pháp:

<tính chất> <kiểu dữ liệu> <tên thuộc tính>;

Kiểu dữ liệu: có thể là các kiểu dữ liệu cơ bản sẵn có của java, có thể là các lớp do người dùng tự định nghĩa.

Tên thuộc tính: được đặt tên theo quy tắc đặt tên biến của java.

Tính chất: Các thuộc tính và phương thức của lớp có các tính chất được đặc trưng bởi các từ khoá sau (giá trị mặc định là **public**):

public: có thể được truy cập từ bên ngoài lớp định nghĩa.

protected: chỉ được truy cập từ lớp định nghĩa và các lớp kế thừa từ lớp đó.

private: chỉ được truy cập trong phạm vi bản thân lớp định nghĩa.

static: được dùng chung cho một thể hiện của lớp, có thể được truy cập trực tiếp bằng <tên lớp>.<tên thuộc tính> mà không cần khởi tạo một thể hiện của lớp.

abstract: định nghĩa một thuộc tính trừu tượng. Thuộc tính này không thể truy nhập trong lớp nhưng có thể bị định nghĩa chồng ở các lớp kế thừa.

final: một thuộc tính hằng, không bị định nghĩa chồng ở các lớp kế thừa.

native: dùng cho phương thức khi cài đặt phụ thuộc môi trường trong một ngôn ngữ khác, như C hay hợp ngữ.

synchronized: dùng cho phương thức tới hạn, nhằm ngăn các tác động của các đối tượng khác khi phương thức đang được thực hiện.

Chương trình 4.2 minh hoạ việc khai báo hai thuộc tính là tên và tuổi của lớp Người (Person).

Chương trình 4.2

```
package vidu.chuong4;  
class Person  
{
```

```
public String name;  
public int age;  
}
```

Khai báo phương thức của lớp

Phương thức của lớp được khai báo theo cú pháp

```
<tính chất> <kiểu trả về> <tên phương thức> ([<các tham số>])  
    [throws <các ngoại lệ>]  
{  
}
```

Tính chất: đặc trưng bởi các từ khoá tương tự như tính chất của thuộc tính. Giá trị mặc định là **public**.

Kiểu trả về: Kiểu dữ liệu trả về của phương thức, có thể là kiểu dữ liệu sẵn có của java hoặc là kiểu do người dùng tự định nghĩa.

Tên phương thức: tuân theo qui tắc đặt tên biến của java.

Các ngoại lệ: là một đối tượng đặc biệt được tạo ra khi chương trình gặp lỗi. Java sẽ trả lại cho chương trình ngoại lệ này theo từ khoá **throws**. Các ngoại lệ, nếu có, được phân cách nhau bởi dấu phẩy.

Các tham số: các tham số của phương thức, được liệt kê theo cặp <kiểu tham số> <tên tham số>, các tham số được phân biệt bởi dấu phẩy.

Chương trình 4.3 mô tả việc khai báo phương thức show() để hiển thị thông tin cá nhân của lớp Person.

Chương trình 4.3

```
package vidu.chuong4;  
class Person  
{  
    public String name;  
    public int age;  
  
    public void show()  
    {  
        System.out.println( name + " is " + age + " years old!");  
    }  
}
```

Phương thức khởi tạo của lớp

Phương thức khởi tạo (Constructor) được dùng để khởi tạo một thể hiện cụ thể của một lớp, nghĩa là gán các giá trị khởi đầu cho các thuộc tính, nếu có, và tạo ra một đối tượng cụ thể. Phương thức khởi tạo phải cùng tên với lớp.

Lưu ý:

Phương thức khởi tạo phải có tên trùng với tên của lớp

Phương thức khởi tạo không có giá trị trả về

Phương thức khởi tạo có tính chất public

Có thể có nhiều phương thức khởi tạo của cùng một lớp

Chương trình 4.4a minh họa một phương thức khởi tạo của lớp Person bằng cách gán giá trị cho các thuộc tính tên và tuổi.

Chương trình 4.4a

```
package vidu.chuong4;
class Person
{
    public      name;
    String      age;
    public int

    // Phương thức khởi dựng
    public Person(String name1, int age1)
    {
        name = name1;
        age = age1;
    }

    public void show()
    {
        System.out.println( name + " is " + age + " years old!");
    }
}
```

Chương trình 4.4b minh họa cách dùng lớp Person mà chúng ta vừa định nghĩa trong chương trình 4.4a. Chương trình này sẽ tạo ra một đối tượng myPerson của lớp Person với 68

các thuộc tính có giá trị khởi tạo: name = "Minh" và age = "21". Sau đó, chương trình sử dụng phương thức show() của đối tượng myPerson để in ra dòng thông báo "Minh is 21 years old!".

Chương trình 4.4b

```
package vidu.chuong4;
class PersonDemo
{
    public static void main(String args[])
    {
        Person myPerson = new Person("Minh", 21);
        myPerson.show();
    }
}
```

Biến this

Biến **this** là một biến ẩn đặc biệt luôn tồn tại trong các lớp java: một lớp có đúng một biến ẩn **this**. Biến này được sử dụng trong khi chạy và nó trỏ đến bản thân lớp chứa nó. Biến **this** thường được sử dụng trong các hàm khởi tạo của lớp.

Chương trình 4.4c khai báo một lớp hoàn toàn giống với lớp được khai báo trong chương trình 4.4a, nhưng chỉ khác là có dùng biến **this** trong hàm khởi tạo của lớp.

Chương trình 4.4c

```
package vidu.chuong4;
class Person
{
    public String  name;
    public int     age;

    // Phương thức khởi dựng
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
```

```

public void show()
{
    System.out.println( name + " is " + age + " years old!");
}
}

```

Trong chương trình 4.4c, ta chú ý đến hàm khởi tạo của lớp, hàm này có hai biến cục bộ là name và age, trùng với các biến của lớp. Do đó, trong phạm vi hàm này, biến this.name và this.age sẽ chỉ các biến của lớp, còn các biến name và age sẽ chỉ các biến cục bộ của hàm. Cho nên, các lệnh gán vẫn thực thi như trong chương trình 4.4a.

4.1.2 Sự kế thừa

Sự kế thừa được sử dụng khi muốn tạo một lớp mới từ một lớp đã biết. Khi đó, tất cả các thuộc tính và phương thức của lớp cũ đều trở thành thuộc tính và phương thức của lớp mới. Lớp cũ được gọi là lớp cha, lớp mới được gọi là lớp con.

Khai báo lớp kế thừa

Khai báo lớp kế thừa được thực hiện bởi từ khoá **extends**:

```

<thuộc tính> <tên lớp con> extends <tên lớp cha>
{
}

```

Chương trình 4.5a minh hoạ việc tạo một lớp Nhân viên (Employee) được kế thừa từ lớp Person đã được xây dựng trong phần 4.1.1.

Chương trình 4.5a

```

package vidu.chuong4;
class Employee extends Person
{
    public float salary;

    // Phương thức khởi dựng
    public Employee(String name, int age, float salary)
    {
        super(name, age);
        this.salary = salary;
    }
}

```

Khi đó, đoạn chương trình của chương trình 4.5b vẫn in ra dòng thông báo “Minh is 21 years old!” vì khi đó đối tượng myEmployee gọi đến phương thức show() được kế thừa từ lớp Person.

Chương trình 4.5b

```
package vidu.chuong4;
class EmployeeDemo1
{
    public static void main(String args[])
    {
        Employee myEmployee = new Employee(“Minh”, 21,
        300f); myEmployee.show();
    }
}
```

Khai báo phương thức nạp chồng

Khi muốn thay đổi nội dung của các phương thức được kế thừa từ lớp cha, ta dùng cách khai báo phương thức nạp chồng. Thực ra là khai báo lại một phương thức mới có cùng tên và kiểu với một phương thức đã có trong lớp cha.

Chương trình 4.6a sẽ khai báo nạp chồng phương thức show() của lớp Employee mà không dùng lại phương thức show() của lớp Person nữa.

Chương trình 4.6a

```
package vidu.chuong4;
class Employee extends Person
{
    public float salary;

    // Phương thức khởi dựng
    public Employee(String name, int age, float salary)
    {
        super(name, age);
        this.salary = salary;
    }
}
```

```

// Khai báo nạp chồng
public void show()
{
    System.out.println( name + “has a salary of”
        salary + “$/month”);
}
}

```

Khi đó, đoạn chương trình 4.6b sẽ in ra dòng thông báo “Minh has a salary of 300\$/month” thay vì dòng thông báo “Minh is 21 years old!” như trong chương trình 4.5b. Lí do là lúc này, đối tượng myEmployee sẽ gọi phương thức show() của lớp Employee mà không gọi phương thức show() của lớp Person nữa.

Chương trình 4.6b

```

package vidu.chuong4;
class EmployeeDemo2
{
    public static void main(String args[])
    {
        Employee myEmployee = new Employee(“Minh”, 21,
            300f); myEmployee.show();
    }
}

```

Quy tắc truy nhập trong kế thừa

Các quy tắc này quy định khả năng truy nhập của lớp con đối với các thuộc tính và phương thức của lớp cha:

private: chỉ được truy nhập trong phạm vi lớp cha, lớp con không truy nhập được. Tất cả các lớp ngoài lớp cha đều không truy nhập được.

protected: lớp con có thể truy nhập được. Tất cả các lớp không kế thừa từ lớp cha đều không truy nhập được.

final: lớp con có thể sử dụng được nhưng không thể khai báo nạp chồng được.

public: lớp con có thể sử dụng và nạp chồng được. Tất cả các lớp bên ngoài đều sử dụng được.

4.2 KẾ THỪA BỘI

Nhằm tránh những nhập nhằng của tính chất đa kế thừa của C++, Java không cho phép kế thừa trực tiếp từ nhiều hơn một lớp cha. Nghĩa là Java không cho phép đa kế thừa trực tiếp, nhưng cho phép cài đặt nhiều giao tiếp (Interface) để có thể thừa hưởng thêm các thuộc tính và phương thức của các giao tiếp đó.

4.2.1 Giao tiếp

Khai báo giao tiếp

Cú pháp khai báo một giao tiếp như sau:

```
[public] interface <tên giao tiếp> [extends <danh sách giao tiếp>]
{
}
```

Tính chất: tính chất của một giao tiếp luôn là **public**. Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**.

Tên giao tiếp: tuân thủ theo quy tắc đặt tên biến của java.

Danh sách các giao tiếp: danh sách các giao tiếp cha đã được định nghĩa để kế thừa, các giao tiếp cha được phân cách nhau bởi dấu phẩy. (Phần trong ngoặc vuông “[]” là tùy chọn).

Lưu ý:

Một giao tiếp chỉ có thể kế thừa từ các giao tiếp khác mà không thể được kế thừa từ các lớp sẵn có.

Chương trình 4.7 minh họa việc khai báo một giao tiếp không kế thừa từ bất kì một giao tiếp nào.

Chương trình 4.7

```
package vidu.chuong4;
public interface Product
{
}
```

Khai báo phương thức của giao tiếp

Cú pháp khai báo một phương thức của giao tiếp như sau:

```
[public] <kiểu giá trị trả về> <tên phương thức> ([<các tham số>])
    [throws <danh sách ngoại lệ>];
```

Tính chất: tính chất của một thuộc tính hay phương thức của giao tiếp luôn là **public**.

Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**. Đối với thuộc tính, thì chắc chắn luôn phải thêm là hằng (**final**) và tĩnh (**static**).

Kiểu giá trị trả về: có thể là các kiểu cơ bản của java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).

Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp

Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.

Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

Lưu ý:

Các phương thức của giao tiếp chỉ được khai báo dưới dạng mẫu mà không có cài đặt chi tiết (có dấu chấm phẩy ngay sau khai báo và không có phần cài đặt trong dấu “{}”). Phần cài đặt chi tiết của các phương thức chỉ được thực hiện trong các lớp (class) sử dụng giao tiếp đó.

Các thuộc tính của giao tiếp luôn có tính chất là hằng (final), tĩnh (static) và public. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính của giao tiếp.

Chương trình 4.8 minh họa việc khai báo một thuộc tính và một phương thức của giao tiếp Product đã được khai báo trong chương trình 4.7: thuộc tính lưu nhãn hiệu của nhà sản xuất sản phẩm; phương thức dùng để truy xuất giá bán của sản phẩm.

Chương trình 4.8

```
package vidu.chuong4;
public interface Product
{
    public static final String MARK = “Adidas”;
    public float getCost();
}
```

4.2.2 Sử dụng giao tiếp

Vì giao tiếp chỉ được khai báo dưới dạng các phương thức mẫu và các thuộc tính hằng nên việc sử dụng giao tiếp phải thông qua một lớp có cài đặt giao tiếp đó. Việc khai báo một lớp có cài đặt giao tiếp được thực hiện thông qua từ khóa **implements** như sau:

```
<tính chất> class <tên lớp> implements <các giao tiếp>
{
```

Tính chất và tên lớp được sử dụng như trong khai báo lớp thông thường.

Các giao tiếp: một lớp có thể cài đặt nhiều giao tiếp. Các giao tiếp được phân cách nhau bởi dấu phẩy. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các giao tiếp mà nó sử dụng.

Lưu ý:

Một phương thức được khai báo trong giao tiếp phải được cài đặt cụ thể trong lớp có cài đặt giao tiếp nhưng không được phép khai báo chồng. Nghĩa là số lượng các tham số của phương thức trong giao tiếp phải được giữ nguyên khi cài đặt cụ thể trong lớp.

Chương trình 4.9 minh họa việc cài đặt một lớp giày (Shoe) cài đặt giao tiếp Product với các thuộc tính và phương thức đã được khai báo trong chương trình 4.8.

Chương trình 4.9

```
package vidu.chuong4;

public class Shoe implements Product
{
    // Cài đặt phương thức được khai báo trong giao tiếp
    public float getCost()
    {
        return 10f;
    }

    // Phương thức truy nhập nhãn hiệu sản phẩm
    public String getMark()
    {
        return MARK;
    }

    // Phương thức main
    public static void main(String args[])
    {
        Shoe myShoe = new Shoe();
        System.out.println("This shoe is " +
                           myShoe.getMark() + " having a cost
                           of $" + myShoe.getCost());
    }
}
```

Chương trình 4.9 sẽ in ra dòng: “This shoe is Adidas having a cost of \$10”. Hàm getMark() sẽ trả về nhãn hiệu của sản phẩm, là thuộc tính đã được khai báo trong giao tiếp. Hàm getCost() là cài đặt riêng của lớp Shoe đối với phương thức đã được khai báo trong giao tiếp Product mà nó sử dụng, cài đặt này trả về giá trị 10 đối với lớp Shoe.

4.3 LỚP TRỪU TƯỢNG

Lớp trừu tượng là một dạng lớp đặc biệt, trong đó các phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết. Việc cài đặt chi tiết các phương thức chỉ được thực hiện ở các lớp con kế thừa lớp trừu tượng đó.

Lớp trừu tượng được sử dụng khi muốn định nghĩa một lớp mà không thể biết và định nghĩa ngay được các thuộc tính và phương thức của nó.

4.3.1 Khai báo

Khai báo lớp trừu tượng

Lớp trừu tượng được khai báo như cách khai báo các lớp thông thường, ngoại trừ có thêm từ khoá **abstract** trong phần tính chất:

```
[public] abstract class <tên lớp>
{
}
```

Tính chất: mặc định là **public**, bắt buộc phải có từ khoá **abstract** để xác định đây là một lớp trừu tượng.

Tên lớp: tuân thủ theo quy tắc đặt tên lớp thông thường của java.

Lưu ý:

Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng. (Khai báo kế thừa thông qua từ khoá **extends** như khai báo kế thừa thông thường).

Chương trình 4.10 khai báo một lớp trừu tượng là lớp động vật (Animal) một cách chung chung.

Chương trình 4.10

```
package vidu.chuong4;
abstract class Animal
{
}
```

Khai báo phương thức của lớp trừu tượng

Tất cả các thuộc tính và phương thức của lớp trừu tượng đều phải khai báo là trừu tượng. Hơn nữa, các phương thức của lớp trừu tượng chỉ được khai báo ở dạng khuôn mẫu mà không có phần khai báo chi tiết. Cú pháp khai báo phương thức của lớp trừu tượng:

[public] abstract <kiểu dữ liệu trả về> <tên phương thức>

([<các tham số>]) [throws <các ngoại lệ>];

Tính chất: tính chất của một thuộc tính hay phương thức của lớp trừu tượng luôn là **public**. Nếu không khai báo tường minh thì giá trị mặc định cũng là **public**.

Kiểu dữ liệu trả về: có thể là các kiểu cơ bản của java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).

Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp

Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.

Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

Lưu ý:

Tính chất của phương thức trừu tượng không được là **private** hay **static**. Vì phương thức trừu tượng chỉ được khai báo chi tiết (nạp chồng) trong các lớp dẫn xuất (lớp kế thừa) của lớp trừu tượng. Do đó, nếu phương thức là **private** thì không thể nạp chồng, nếu phương thức là **static** thì không thể thay đổi trong lớp dẫn xuất.

Phương thức trừu tượng chỉ được khai báo dưới dạng khuôn mẫu nên không có phần dấu móc “{}” mà kết thúc bằng dấu chấm phẩy “;”.

Chương trình 4.11 khai báo hai phương thức của lớp trừu tượng Animal trong chương trình 4.10: Phương thức getName() trả về tên loài động vật, dù chưa biết tên cụ thể loài nào nhưng kiểu trả về là String. Phương thức getFeet() trả về số chân của loài động vật, cũng chưa biết cụ thể là bao nhiêu chân nhưng kiểu trả về là int.

Chương trình 4.11

```
package vidu.chuong4;

abstract class Animal
{
    abstract String getName();
    abstract int getFeet();
}
```

```
}
```

4.3.2 Sử dụng lớp trừu tượng

Lớp trừu tượng được sử dụng thông qua các lớp dẫn xuất của nó. Vì chỉ có các lớp dẫn xuất mới cài đặt cụ thể các phương thức được khai báo trong lớp trừu tượng.

Chương trình 4.12a khai báo lớp về loài chim (Bird) kế thừa từ lớp Animal trong chương trình 4.11. Lớp này cài đặt chi tiết hai phương thức đã được khai báo trong lớp Animal: phương thức getName() sẽ trả về tên loài là “Bird”; phương thức getFeet() trả về số chân của loài chim là 2.

Chương trình 4.12a

```
package vidu.chuong4;
public class Bird extends Animal
{
    // Trả về tên loài chim
    public String getName()
    {
        return “Bird”;
    }

    // Trả về số chân của loài chim
    public int getFeet()
    {
        return 2;
    }
}
```

Chương trình 4.12b khai báo lớp về loài mèo (Cat) cũng kế thừa từ lớp Animal trong chương trình 4.11. Lớp này cài đặt chi tiết hai phương thức đã được khai báo trong lớp Animal: phương thức getName() sẽ trả về tên loài là “Cat”; phương thức getFeet() trả về số chân của loài mèo là 4.

Chương trình 4.12b

```
package vidu.chuong4;
public class Cat extends Animal
```

```

{
    // Trả về tên loài mèo
    public String getName()
    {
        return "Cat";
    }

    // Trả về số chân của loài mèo
    public int getFeet()
    {
        return 4;
    }
}

```

Chương trình 4.12c sử dụng lại hai lớp Bird và Cat trong các chương trình 4.12a và 4.12b. Chương trình này sẽ hiển thị hai dòng thông báo:

The Bird has 2 feets

The Cat has 4 feets

Chương trình 4.12c

```

package vidu.chuong4;
public class AnimalDemo
{
    public static void main(String args[])
    {
        Bird myBird = new Bird();
        System.out.println("The " + myBird.getName() + "
                           has " + myBird.getFeet() + " feets");
        Cat myCat = new Cat();
        System.out.println("The " + myCat.getName() + "
                           has " + myCat.getFeet() + " feets");
    }
}

```

4.4.1 Nạp chồng

Java cho phép trong cùng một lớp, có thể khai báo nhiều phương thức có cùng tên. Nạp chồng là hiện tượng các phương thức có cùng tên. Có hai kiểu nạp chồng trong Java:

Các phương thức của cùng một lớp có cùng tên. Khi hai phương thức của một lớp có cùng tên thì bắt buộc chúng phải có:

Hoặc danh sách các tham số khác nhau

Hoặc kiểu trả về khác nhau

Hoặc kết hợp hai điều kiện trên.

Nếu không, java sẽ không phân biệt được chúng. Ví dụ nếu trong cùng một lớp:

```
// Chấp nhận được
public int add(int x, int y){...}
public float add(float x, int y){...}
```

Không chấp nhận được

```
public int add(int x, int
y){...} public int add(int x,
int y){...}
```

Phương thức của lớp con có cùng tên với phương thức của lớp cha. Trong trường hợp này, các phương thức nạp chồng có thể có cùng danh sách tham số và có cùng kiểu trả về (xem lại phần khai báo phương thức nạp chồng trong mục kế thừa 4.1.2).

4.4.2 Đa hình

Đa hình là việc triệu gọi đến các phương thức nạp chồng của đối tượng. Khi một phương thức nạp chồng được gọi, chương trình sẽ dựa vào kiểu các tham số và kiểu trả về để gọi phương thức của đối tượng cho phù hợp.

Chương trình 4.13 minh họa việc khai báo nhiều hàm add() để cộng hai số hoặc cộng hai xâu kí tự.

Chương trình 4.13

```
package vidu.chuong4;
public class Operator
{
    // Cộng hai số nguyên
    public int add(int x, int y)
    {
        return (x + y);
    }
}
```



```

    }

    // Cộng hai số thực
    public float add(float x, float y)
    {
        return (x + y);
    }

    // Cộng hai chuỗi kí tự
    public String add(String a, String b)
    {
        return (a + b);
    }

    // Hàm main
    public static void main(String args[])
    {
        Operator myOperator = new Operator();
        System.out.println("The (5+19) is " + myOperator.add(5, 19));
        System.out.println("The ('ab' + 'cd') is '" +
            myOperator.add("ab", "cd") + "'");
    }
}

```

Chương trình 4.13 sẽ hiển thị ra hai dòng thông báo:

The (5+19) is 24

The ('ab' + 'cd') is 'abcd'

Trong lớp Operator có hai phương thức cùng tên và cùng có hai tham số đầu vào là add(). Khi chương trình thực thi lệnh myOperator.add(5, 19), chương trình sẽ tự đối chiếu các kiểu tham số, thấy 5 và 19 có dạng gần với kiểu int nhất, nên phương thức add(int, int) sẽ được gọi và trả về giá trị là 24.

Khi chương trình thực thi lệnh myOperator.add("ab", "cd"), chương trình sẽ tự đối chiếu các kiểu tham số, thấy 'ab' và 'cd' có dạng gần với kiểu String nhất, nên phương thức add(String, String) sẽ được gọi và trả về giá trị là 'abcd'.

Lưu ý:

Khi gọi hàm với các kiểu dữ liệu khác với các hàm đã được khai báo, sẽ có sự chuyển đổi kiểu ngầm định diễn ra. Khi không thể thực hiện chuyển đổi kiểu ngầm định, java sẽ phát sinh một thông báo lỗi.

Chẳng hạn, trong chương trình 4.13, nếu ta thực thi lệnh `myOperator.add(4.0f, 5)` có dạng `add(float, int)`, chương trình sẽ chuyển ngầm định số nguyên 5 thành float (chuyển từ kiểu `int` sang float thuộc diện nói rộng kiểu, là kiểu chuyển ngầm định trong java) để có thể sử dụng dạng được khai báo `add(float, float)` và kết quả sẽ là 9.0f.

Nếu ta thực thi lệnh `myOperator.add('ab', 5)` có dạng `add(String, int)`, vì `int` không thể chuyển ngầm định thành `String` nên lệnh này sẽ phát sinh lỗi. Để tránh lỗi này, phải chuyển đổi kiểu tường minh cho số 5 thành kiểu `String` bằng một trong các cách sau:

```
myOperator.add("ab", (new Int(5)).toString());  
myOperator.add("ab", 5 + "");
```

4.5 CASE STUDY II

Trong phần này, chúng ta sẽ viết một chương trình quản lý nhân viên của một công ty. Bao gồm các lớp chính:

Lớp `Human` là một lớp trừu tượng, chỉ có một phương thức duy nhất là `show()`.

Lớp `Person` là lớp kế thừa từ lớp `Human`, có hai thuộc tính là tên (`name`) và tuổi (`age`). Để đóng gói dữ liệu các thuộc tính này có dạng `private` và các phương thức truy nhập chúng (`get` và `set`). Ngoài ra lớp này còn cài đặt phương thức `show()` kế thừa từ lớp trừu tượng `Human`.

Lớp `Employee` là lớp kế thừa từ lớp `Person`, có thêm thuộc tính là lương (`salary`). Thuộc tính này cũng có dạng `private` để đóng gói dữ liệu và cần các phương thức truy nhập `get/set`. Lớp này cài đặt lại phương thức `show()`. Hơn nữa, lớp `Employee` còn có thêm hai phương thức `addSalary()` và `addSalary(float)` để tính tăng lương cho nhân viên: một phương thức tăng lương theo tỉ lệ mặc định là 10% (không cần tham số), và một phương thức tăng theo giá trị cụ thể đưa vào (cần tham số).

Các phần tiếp theo sẽ trình bày phân cài đặt chi tiết cho các lớp này.

4.5.1 Lớp Human

Lớp `Human` là một lớp trừu tượng, chỉ có một phương thức duy nhất là `show()`. Đây là nội dung tập tin `Human.java`.

Chương trình 4.14a

```
package vidu.chuong4;  
abstract class Human  
{  
    abstract void show();
```

```
}
```

4.5.2 Lớp Person

Lớp Person là lớp kế thừa từ lớp Human:

Có hai thuộc tính là tên (name) và tuổi (age) có dạng private

Các phương thức truy nhập các thuộc tính name (getName() và setName(String)) và age (getAge() và setAge(int)).

Cài đặt chồng phương thức show() kế thừa từ lớp trừu tượng Human.

Đây là nội dung tập tin Person.java.

Chương trình 4.14b

```
package vidu.chuong4;
class Person extends Human
{
    private String name;
    private int age;

    // Phương thức khởi dựng không có tham số
    public Person()
    {
        super();
        name = "";
        age = 0;
    }

    // Phương thức khởi dựng có tham số
    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    /* Phương thức truy nhập thuộc tính name */
    public String getName()
    {
```

```

        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    /* Phương thức truy nhập thuộc tính age */
    public int getAge()
    {
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    // Khai báo nạp chồng
    public void show()
    {
        System.out.println( name + " is " + age + " years old!");
    }
}

```

4.5.3 Lớp Employee

Lớp Employee là lớp kế thừa từ lớp Person:

Có thêm thuộc tính là lương (salary) cũng có dạng private để đóng gói dữ liệu và cần các phương thức truy nhập get/set.

Lớp này cài đặt lại phương thức show().

Có thêm hai phương thức addSalary() và addSalary(float) để tính tăng lương cho nhân viên: phương thức addSalary() tăng lương theo tỉ lệ mặc định là 10% (không cần tham số), phương thức addSalary(float) tăng theo giá trị cụ thể đưa vào (cần tham số).

Sau đây là nội dung tập tin Employee.java

Chương trình 4.14c

```
package vidu.chuong4;
class Employee extends Person
{
    private float salary;

    // Phương thức khởi dựng không có tham số
    public Employee()
    {
        super();
        salary = 0f;
    }

    // Phương thức khởi dựng có tham số
    public Employee(String name, int age, float salary)
    {
        super(name, age);
        this.salary = salary;
    }

    /* Phương thức truy nhập thuộc tính salary */
    public float getSalary()
    {
        return salary;
    }

    public void setSalary(float salary)
    {
        this.salary = salary;
    }

    // Khai báo nạp chồng
    public void show()
    {
        System.out.println( getName() + " is " + getAge()

```

```

        “ years old having a salary of $”
        salary + “/month!”);

    }

    /* Phương thức tăng lương */
    public void addSalary()
    {
        salary = salary*1.1f;
    }

    public void addSalary(float addition)
    {
        salary += addition;
    }
}

```

Lưu ý: Trong phương thức nạp chồng show() của lớp Employee, ta phải dùng các phương thức public được kế thừa từ lớp Person là getName() và getAge() để truy nhập đến thuộc tính name và age mà không thể truy xuất trực tiếp. Lí do là các thuộc tính name và age có dạng private trong lớp Person nên không thể truy xuất trực tiếp trong các lớp dẫn xuất. Do đó, ta phải truy xuất chúng thông qua các phương thức truy nhập public của lớp Person.

4.5.4 Chương trình demo

Chương trình 4.14d chứa hàm main để chạy minh họa việc sử dụng các lớp Person và Employee. Đây là nội dung của tập tin Casestudy2.java

Chương trình 4.14d

```

package vidu.chuong4;

public class Casestudy2
{
    // Hàm main
    public static void main(String args[])
    {
        // Sử dụng lớp Person
        Person myPerson = new Person(“Vinh”, 25);
        myPerson.show();
    }
}

```

```

// Sử dụng lớp Employee
Employee myEmployee = new Employee("Vinh", 25, 300f);
myEmployee.show();
// Tăng lương theo mặc định
myEmployee.addSalary();
myEmployee.show();
// Tăng lương lên $50
myEmployee.addSalary(50f);
myEmployee.show();
}
}

```

Chương trình 4.14d sẽ hiển thị nội dung như sau:

Vinh is 25 years old!

Vinh is 25 years old having a salary of \$300/month!

Vinh is 25 years old having a salary of \$330/month!

Vinh is 25 years old having a salary of \$380/month!

Dòng thứ nhất in ra dòng thông báo theo phương thức show() của lớp Person. Dòng thứ hai cũng hiển thị thông báo theo phương thức show() của lớp Employee đã được khai báo nạp chồng. Dòng thứ ba hiển thị thông báo sau khi đã tăng lương theo mặc định (10%) từ 300\$ lên 330\$. Dòng thứ tư hiển thị thông báo sau khi tăng lương thêm một lần nữa với lượng tăng là 50\$ từ 330\$ lên 380\$.

TỔNG KẾT CHƯƠNG 4

Nội dung chương 4 đã tập trung trình bày các vấn đề cơ bản liên quan đến kỹ thuật lập trình hướng đối tượng trong Java:

Một lớp trong java được khai báo với từ khoá class, với các tính chất có thể có là public, final hoặc abstract.

Khi khai báo một lớp là trừu tượng thì các phương thức của nó cũng được khai báo là trừu tượng và chỉ khai báo dạng khuôn mẫu mà không được khai báo chi tiết.

Không thể tạo ra một đối tượng từ một lớp trừu tượng.

Có thể khai báo một lớp kế thừa từ một lớp khác thông qua từ khoá extends. Một lớp java chỉ có thể được kế thừa từ nhiều nhất một lớp cha mà thôi.

Trong kế thừa, lớp con có thể định nghĩa lại (nạp chồng) các phương thức được kế thừa từ lớp cha.

Một giao tiếp trong java được khai báo thông qua từ khoá interface. Các thuộc tính của giao tiếp phải là thuộc tính hằng (final) và static. Các phương thức chỉ được khai báo dưới dạng khuôn mẫu mà không được cài đặt chi tiết.

Có thể khai báo một giao tiếp kế thừa từ một giao tiếp khác cũng bằng từ khoá extends. Một giao tiếp trong java có thể được kế thừa từ nhiều giao tiếp khác.

Một giao tiếp trong java chỉ được sử dụng thông qua một lớp java cài đặt cụ thể giao tiếp đó. Khi đó, lớp java phải cài đặt chi tiết tất cả các phương thức được khai báo trong giao tiếp.

Trong một lớp java, có thể khai báo nhiều phương thức có cùng tên. Khi thực thi, chương trình sẽ tự động chọn phương thức có khuôn mẫu kiểu tham số trùng với lệnh để thực hiện lệnh tương ứng.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

Trong các khai báo lớp sau, khai báo nào là đúng:

`class myClass`

`Class myClass`

`public class MyClass`

`Public class MyClass`

Trong các khai báo phương thức của một lớp (không trừu tượng) sau, khai báo nào là đúng:

`String getName{return name;}`

`String getName(){return name;}`

`String getName(){return name;};`

`String getName();`

Trong các khai báo phương thức của một lớp trừu tượng sau, khai báo nào là đúng:

`abstract String getName{return name;}`

`abstract String getName(){return name;}`

`abstract String getName(){return name;};`

`abstract String getName();`

Trong các khai báo một giao tiếp sau, khai báo nào là đúng:

`public abstract MyInterface{...}`

`public class MyInterface{...}`

`public interface MyInterface{...}`

`public interface MyInterface();`

Trong các khai báo kế thừa lớp sau, giả sử Aclass và Bclass là các lớp có sẵn, khai báo nào là đúng:

```
public class MyClass extends Aclass {...}
public class MyClass Extends Aclass {...}
public class MyClass extends Aclass, Bclass {...}
public class MyClass extends Aclass, extends Bclass {...}
```

Trong các khai báo kế thừa giao tiếp sau, giả sử Ainterface và Binterface là các giao tiếp có sẵn, khai báo nào là đúng:

```
public interface MyInterface extends Ainterface {...}
public interface MyInterface Extends Ainterface {...}
public interface MyInterface extends Ainterface, Binterface {...}
d. public interface MyInterface extends Ainterface, extends Binterface {...}
```

Trong các khai báo một lớp sử dụng giao tiếp sau, giả sử Ainterface và Binterface là các giao tiếp có sẵn, khai báo nào là đúng:

```
public class MyClass extends Ainterface {...}
public class MyClass implements Ainterface {...}
public class MyClass implements Ainterface, Binterface {...}
public class MyClass implements Ainterface, implements Binterface {...}
```

Xét đoạn chương trình cài đặt một lớp sử dụng một giao tiếp sau:

```
public interface MyInterface{
    public void show(String a);
}
// Khai báo lớp sử dụng giao tiếp
public class MyClass implements MyInterface{
    ...
}
```

Khi đó, lớp MyClass chỉ có duy nhất một phương thức cài đặt chi tiết phương thức show của giao tiếp MyInterface. Trong các cách cài đặt sau, cách nào là đúng:

```
a. public int show(String a){
    System.out.println(a);} b. public void show(){
    System.out.println("Show!");} c. public void
show(String a){ System.out.println(a);}
d. public void show(String a, String b){ System.out.println(a+b);}
```

Xét đoạn chương trình cài đặt phép toán cộng với các số hạng có kiểu khác nhau:

```
public class MyOperator {
    public static int add(int a, int b){return a+b;}
```

```

        public static float add(float a, float b){return a+b;}
        public static double add(double a, double b){return
a+b;} public static String add(String a, String
b){return a+b;}
    }

```

Khi ta gọi lệnh `MyOperator.add(-5, 100)`, chương trình sẽ gọi phương thức nào?

- a. `add(int, int)`
- b. `add(float, float)`
- c. `add(double, double)`
- d. `add(String, String)`
- e. Thông báo lỗi

Cũng với lớp `MyOperator` trong bài 9, khi ta gọi lệnh `MyOperator.add(5.0d, 100f)`, chương trình sẽ gọi phương thức nào?

```

add(int, int)
add(float, float)
add(double, double)
add(String, String)
Thông báo lỗi

```

Cũng với lớp `MyOperator` trong bài 9, khi ta gọi lệnh `MyOperator.add("abcd", 100f)`, chương trình sẽ gọi phương thức nào?

```

add(int, int)
add(float, float)
add(double, double)
add(String, String)
Thông báo lỗi

```

Về sự khác nhau giữa lớp và giao tiếp trong java, những nhận định nào sau đây là đúng:

- Lớp được kế thừa tối đa từ một lớp khác, giao tiếp có thể kế thừa từ nhiều giao tiếp khác.
- Lớp có thể kế thừa từ giao tiếp, nhưng giao tiếp không được kế thừa từ các lớp.
- Thuộc tính của lớp có thể có tính chất bất kì, thuộc tính của giao tiếp phải là hằng số và tĩnh.
- Phương thức của lớp có thể được cài đặt chi tiết, phương thức của giao tiếp chỉ được khai báo ở dạng khuôn mẫu.

Xét đoạn chương trình khai báo hai lớp kế thừa nhau như sau:

```

public class Person {
    public void show(){System.out.println("This is a person!");}
}

```

```
}
```

và:

```
public class Employee extends Person {  
    public void show(int x){  
        System.out.println("This is the employee " + x);  
    }  
}
```

Khi đó, đoạn chương trình sau sẽ hiển thị thông báo nào?

```
public class Demo1 {  
    public static void main(String args[]){  
        Employee myEmployee = new Employee();  
        myEmployee.show();  
    }  
}
```

- a. This is a person!.
- b. This is the employee 0.
- c. Thông báo lỗi.
- d. Không hiển thị gì cả.

Cũng với hai lớp Person và Employee trong bài 14. Đoạn chương trình sau sẽ hiển thị thông báo nào?

```
public class Demo2 {  
    public static void main(String args[]){  
        Employee myEmployee = new Employee();  
        myEmployee.show(10);  
    }  
}
```

- a. This is a person!.
- b. This is the employee 10.
- c. Thông báo lỗi.
- d. Không hiển thị gì cả.

Viết một chương trình khai báo một lớp về các hình chữ nhật Rectangle. Lớp này có hai thuộc tính cục bộ là chiều rộng và chiều dài của hình. Viết hai phương thức khởi dựng tường minh cho lớp này: Một khởi dựng với một tham số kiểu int, khi đó, chiều rộng và chiều dài được thiết lập thành giá trị tham số đưa vào (hình vuông). Một khởi dựng với hai tham số kiểu int tương ứng là chiều rộng và chiều dài.

Viết một giao tiếp khai báo các phép toán cộng hai số, cộng hai xâu, cộng xâu và số. Sau đó viết một lớp cài đặt giao tiếp đó.

Viết một lớp trừu tượng về các hình phẳng, trong đó khai báo các phương thức tính chu vi và diện tích của hình. Sau đó viết ba lớp kế thừa từ lớp trừu tượng đó là: lớp hình vuông, lớp hình chữ nhật và lớp hình tròn. Tự thiết lập các thuộc tính cục bộ cần thiết cho mỗi lớp con và khai báo nạp chồng hai phương thức ban đầu trong mỗi lớp con.

CHƯƠNG 5

BIỂU DIỄN VÀ CÀI ĐẶT

CÁC CẤU TRÚC DỮ LIỆU TRỪU TƯỢNG TRÊN JAVA

Nội dung chương này tập trung trình bày việc cài đặt một số giải thuật và các cấu trúc dữ liệu trừu tượng trên java. Các giải thuật bao gồm:

Phương pháp duyệt và đệ qui

Phương pháp sắp xếp và tìm kiếm

Các cấu trúc dữ liệu trừu tượng bao gồm:

Ngăn xếp và hàng đợi

Danh sách liên kết

Cây nhị phân

Đồ thị

5.1 PHƯƠNG PHÁP DUYỆT VÀ ĐỆ QUI

5.1.1 Các phương pháp duyệt

Các phương pháp duyệt thường xuất hiện trong bài toán tổ hợp nhằm tìm kiếm tất cả các trạng thái (tổ hợp) có thể có của một tập các trạng thái, hoặc tìm ra một tổ hợp thoả mãn tốt nhất một số yêu cầu xác định trong số các trạng thái của tập hợp. Các phương pháp duyệt thông thường bao gồm:

Phương pháp duyệt toàn bộ: Duyệt qua tất cả các trạng thái có thể có của tập hợp.

Phương pháp này phù hợp với bài toán liệt kê và với các tập hợp có số lượng các trạng thái là đủ nhỏ để không tốn thời gian duyệt.

Phương pháp duyệt chọn lọc: Thường áp dụng khi số lượng trạng thái của tổ hợp là rất lớn, không thể sử dụng phương pháp duyệt toàn bộ. Phương pháp này chỉ dùng cho các bài toán tìm kiếm một (hoặc một số) trạng thái tốt nhất (theo một số tiêu chí xác định) của tổ hợp bằng cách chỉ duyệt theo một số trạng thái được cho là có khả năng tìm được kết quả cao nhất. Một số thuật toán duyệt theo phương pháp này thường được áp dụng trong lĩnh vực trí tuệ nhân tạo như thuật toán A*, thuật toán nhánh và biên, thuật toán / .

Ngoài ra, tùy thuộc vào cách thức biểu diễn dữ liệu của trạng thái (cấu trúc dữ liệu), người ta sử dụng các phương pháp duyệt khác nhau:

Duyệt trên mảng

Duyệt trên cây

Duyệt trên đồ thị

Các phương pháp duyệt này sẽ được trình bày trong các mục tương ứng về các đối tượng danh sách tuyến tính (duyệt trên mảng), cây nhị phân (duyệt trên cây), đồ thị (duyệt trên đồ thị).

5.1.2 Phương pháp đệ qui

Phương pháp đệ qui: được định nghĩa theo qui nạp toán học, một đối tượng được biểu diễn qua chính nó với một phạm vi nhỏ hơn.

Ví dụ, định nghĩa số Fibonacci thứ n:

$$F(n) = F(n-1) + F(n-2) \text{ với } n > 2$$

$$F(1) = F(2) = 1$$

là một định nghĩa mang tính đệ qui.

Giải thuật đệ qui: là phương pháp giải bài toán bằng cách rút gọn bài toán thành một (hoặc một số) bài toán con tương tự như vậy nhưng với dữ liệu nhỏ hơn với trạng thái dừng tồn tại.

Ví dụ, thủ tục tính số Fibonacci thứ n:

```
int Fibo(int n){  
    if(n == 1 || n == 2) return 1;  
    return (Fibo(n-1) + Fibo(n-2));  
}
```

là một giải thuật đệ qui. Nó tính số Fibonacci thứ n thông qua việc tính hai số nhỏ hơn trước nó là số thứ n-1 và n-2. Điều kiện dừng là số thứ 1 và số thứ 2 là 1.

5.2 PHƯƠNG PHÁP SẮP XẾP VÀ TÌM KIẾM

5.2.1 Các phương pháp sắp xếp

Hiện nay, có rất nhiều phương pháp sắp xếp khác nhau:

Sắp xếp nổi bọt (bubble sort)

Sắp xếp chèn (insertion sort)

Sắp xếp chọn (selection sort)

Sắp xếp vun đống (heap sort)

Sắp xếp trộn (merge sort)

Sắp xếp nhanh (quick sort)

Nội dung phần này sẽ trình bày việc cài đặt giải thuật sắp xếp nhanh (quick sort). Vì việc sắp xếp thường gắn liền với một mảng các phần tử có thể so sánh được, cho nên giải thuật này sẽ được cài đặt trong lớp mảng các phần tử (Array). Việc cài đặt các giải thuật sắp xếp còn lại được coi như một bài tập của phần này.

Lớp mảng các phần tử Array

Lớp này có thuộc tính là một mảng các phần tử. Các phần tử của mảng có thể có kiểu bất kì, nhưng phải thoả mãn điều kiện là có thể so sánh được, khi đó, mảng có thể sắp xếp được. Trong phần này, ta sẽ cài đặt mảng các phần tử có kiểu int.

```
class Array{  
    private int *elements;  
}
```

Giải thuật sắp xếp nhanh Quick Sort

tưởng của giải thuật này là chọn một phần tử đóng vai trò là khoá chốt (còn gọi là điểm mốc), các phần tử nhỏ hơn khoá chốt sẽ phải chuyển lên đứng trước khoá chốt. Các phần tử lớn hơn khoá chốt thì phải chuyển xuống đứng sau khoá chốt. Các bước cụ thể như sau:

Chọn một phần tử (bất kì) làm khoá chốt.

Đi từ đầu mảng đến cuối mảng, tìm phần tử đầu tiên lớn hơn khoá chốt, đánh dấu nó là phần tử thứ i.

Đi từ cuối mảng lên đầu mảng, tìm phần tử đầu tiên nhỏ hơn khoá chốt, đánh dấu nó là phần tử thứ j.

Nếu $i < j$, đổi chỗ phần tử thứ i và thứ j.

Sau đó, đi tiếp theo hai chiều, và đổi chỗ, nếu có, cho đến khi $i = j$.

Đổi chỗ phần tử thứ i (cũng là j vào thời điểm này) với phần tử chốt. Khi đó, phần tử chốt là đúng vị trí của nó trong mảng sắp xếp.

Lặp lại giải thuật trên với hai đoạn của mảng: đoạn trước chốt và đoạn sau chốt.

Quá trình sẽ dừng khi mỗi đoạn chỉ còn hai phần tử.

Chương trình 5.1a cài đặt thủ tục sắp xếp nhanh của lớp Array.

Chương trình 5.1a

```
package vidu.chuong5;  
  
class Array{  
    private int[] elements;  
  
    /* Phương thức truy nhập các phần tử của mảng */  
    public int[] get(){  
        return elements;  
    }  
  
    public void set(int[] elements){  
        this.elements = elements;  
    }  
}
```

```

    }

    /* Phương thức sắp xếp */
    public void sort(){
        quick(0, elements.length-1);
    }

    /* Phương thức sắp xếp nhanh */
    private void quick(int left, int right){
        int i=left, j=right;
        int pivot=(left+right)/2, tmp;
        do{
            while(elements[i]<elements[pivot] && i<right)i++; // Quét xuôi
            while(elements[j]>elements[pivot] && j>left)j++; // Quét ngược
            if(i<=j){ // Đổi chỗ hai phần tử
                tmp = elements[i];
                elements[i] = elements[j];
                elements[j] = tmp;
            }
        }while (i<=j);
        if(left < j)    quick(left, j); // Sắp xếp đoạn trước chốt
        if(i < right) quick(i, right); // Sắp xếp đoạn sau chốt
    }
}

```

5.2.2 Các phương pháp tìm kiếm

Phương pháp tìm kiếm sẽ trả về chỉ số của một phần tử nếu nó có mặt trong mảng tìm kiếm, trả về -1 nếu không có phần tử đó trong mảng. Các phương pháp tìm kiếm cơ bản bao gồm:

 Tìm kiếm tuần tự (tuyến tính)

 Tìm kiếm nhị phân

 Tìm kiếm trên cây

Nội dung phần này sẽ trình bày phương pháp tìm kiếm nhị phân. Các phương pháp còn lại được coi như là bài tập của phần này. Phương pháp tìm kiếm nhị phân được thực hiện trên mảng đã sắp xếp. Các bước tiến hành như sau:

Lấy khoá cần tìm so sánh với phần tử ở giữa mảng đã sắp xếp. Nếu bằng, kết thúc tìm kiếm.

Nếu nhỏ hơn, tìm kiếm khoá đó trong nửa đầu của mảng (vẫn theo kiểu nhị phân).

Nếu lớn hơn, tìm kiếm khoá đó trong nửa sau của mảng (vẫn theo kiểu nhị phân).

Quá trình kết thúc khi khoá bằng phần tử giữa mảng, hoặc các đoạn chỉ còn một phần tử.

Chương trình 5.1b cài đặt thủ tục tìm kiếm nhị phân trên lớp mảng Array với các phần tử có kiểu int (khoá tìm kiếm cũng có kiểu int).

Chương trình 5.1b

```
package vidu.chuong5;
class Array{
    private int[] elements;

    /* Phương thức truy nhập các phần tử của mảng */
    public int[] get(){
        return elements;
    }
    public void set(int[] elements){
        this.elements = elements;
    }

    /* Phương thức tìm kiếm */
    public int search(int key){
        quick(0, elements.length-1);    // Sắp xếp mảng, dùng quicksort
        int low=0, hight=elements.length-1, mid;
        while(low <= hight){
            mid = (low + hight)/2;        // Chỉ số giữa
            if(key > elements[mid])
                low = mid+1;              // Tìm nửa sau
            else if(key < elements[mid])
                hight= mid-1;             // Tìm nửa đầu
            else return mid;              // Tìm thấy
        }
        return -1;                       // Không tìm thấy
    }
}
```

```

/* Phương thức sắp xếp */
public void sort(){
    quick(0, elements.length-1);
}

/* Phương thức sắp xếp nhanh */
private void quick(int left, int right){
    int i=left, j=right;
    int pivot=(left+right)/2, tmp;
    do{
        while(elements[i]<elements[pivot] && i<right)i++; // Quét xuôi
        while(elements[j]>elements[pivot] && j>left)j++; // Quét ngược
        if(i<=j){ // Đổi chỗ hai phần tử
            tmp = elements[i];
            elements[i] = elements[j];
            elements[j] = tmp;
        }
    }while (i<=j);
    if(left < j)    quick(left, // Sắp xếp đoạn trước
j);              chót // Sắp xếp đoạn sau chót
    if(i < right) quick(i, right);
}
}

```

5.3 NGĂN XẾP VÀ HÀNG ĐỢI

5.3.1 Ngăn xếp

Ngăn xếp (stack) có các thuộc tính cục bộ:

Mảng lưu các nút của ngăn xếp

Thêm vào một nút

Lấy ra một nút

Định nghĩa một nút

Đơn giản, ta chỉ định nghĩa một nút có giá trị kiểu int:

Chương trình 5.2a

```
package vidu.chuong5;

public class Node{
    private int value;

    /* Các phương thức khởi dựng */
    public Node(){
        value = 0;
    }

    public Node(int value){
        this.value = value;
    }

    /* Phương thức truy nhập thuộc tính value */
    public int getValue(){
        return value;
    }

    public void setValue(int value){
        this.value = value;
    }
}
```

Cài đặt ngăn xếp

Ta coi đỉnh ngăn xếp là cuối mảng lưu giữ các nút. Do đó, các thao tác thêm vào và lấy ra sẽ thêm vào cuối mảng hoặc lấy nút ở cuối mảng ra.

Mảng các giá trị được khai báo động để tiết kiệm bộ nhớ.

Chương trình 5.2b

```
package vidu.chuong5;

public class MyStack{
    private Node[] values;

    /* Các phương thức khởi dựng */
```

```

public MyStack(){

public MyStack(Node[] values){
    this.values = values;
}

/* Phương thức lấy ra một node từ stack */
public Node pop(){
    Node result = null;
    if((values != null)&&(values.length > 0)){
        result = values[values.length - 1];
        // Loại bỏ node cuối cùng
        Node[] tmpNode = new Node[values.length - 1];
        for(int i=0; i<values.length - 1; i++)
            tmpNode[i] = values[i];
        this.values = tmpNode;
    }
    return result;
}

/* Phương thức thêm một node vào stack */
public void push(Node node){
    if(values == null){ // Ngăn xếp đang rỗng
        values = new Node[1];
        values[0] = node;
    }else{ // Ngăn xếp đã có dữ liệu
        Node[] tmpNode = new Node[values.length + 1];
        for(int i=0; i<values.length; i++)
            tmpNode[i] = values[i];
        tmpNode[values.length] = node;
        this.values = tmpNode;
    }
}
}

```

5.3.2 Hàng đợi

Hàng đợi (queue) có các thuộc tính cục bộ:

Mảng các giá trị trong hàng đợi

Các thao tác với hàng đợi:

Thêm vào một nút vào cuối hàng đợi

Lấy ra một nút từ đầu hàng đợi

Chương trình 5.3 cài đặt lớp hàng đợi.

Chương trình 5.3

```
package vidu.chuong5;

public class MyQueue{
    private Node[] values;

    /* Các phương thức khởi dựng */
    public MyQueue(){ }

    public MyQueue(Node[] values){
        this.values = values;
    }

    /* Phương thức lấy ra một node từ đầu queue */
    public Node remove(){
        Node result = null;
        if((values != null)&&(values.length > 0)){
            result = values[0];
            // Loại bỏ node đầu hàng đợi
            Node[] tmpNode = new Node[values.length - 1];
            for(int i=0; i<values.length - 1; i++){
                tmpNode[i] = values[i+1];
            }
            this.values = tmpNode;
        }
        return result;
    }

    /* Phương thức thêm một node vào cuối queue */
    public void insert(Node node){
        if(values == null){ // Hàng đợi đang rỗng
            values = new Node[1];
```

```

        values[0] = node;
    }else{        // Hàng đợi đã có dữ liệu
        Node[] tmpNode = new Node[values.length + 1];
        for(int i=0; i<values.length; i++)
            tmpNode[i] = values[i];
        tmpNode[values.length] = node;
        this.values = tmpNode;
    }
}
}

```

5.4 DANH SÁCH LIÊN KẾT

Nội dung phần này tập trung cài đặt hai loại danh sách liên kết cơ bản:

Danh sách liên kết đơn

Danh sách liên kết kép

5.4.1 Danh sách liên kết đơn

Định nghĩa một nút của danh sách liên kết đơn

Một nút của danh sách liên kết đơn bao gồm:

Giá trị của nút, có dạng là một đối tượng kiểu Node đã được định nghĩa trong chương trình 5.2a

Nút tiếp theo của nút đó.

Một nút của danh sách liên kết đơn được cài đặt trong chương trình 5.4a.

Chương trình 5.4a

```

package vidu.chuong5;
public class SimpleNode{
    private Node value; // Giá trị của node là một đối tượng kiểu
    Node private SimpleNode next; // Node tiếp theo của danh sách
    liên kết

    /* Các phương thức khởi dựng */
    public SimpleNode(){
        value = new Node();
        next = null;
    }

    public SimpleNode(Node value){

```

```

        this.value = value;
        next = null;
    }

    /* Phương thức truy nhập thuộc tính value */
    public Node getValue(){
        return value;
    }
    public void setValue(Node value){
        this.value = value;
    }

    /* Phương thức truy nhập thuộc tính next */
    public SimpleNode getNext(){
        return next;
    }
    public void setNext(SimpleNode next){
        this.next = next;
    }
}

```

Định nghĩa đỉnh tiêu đề của danh sách liên kết đơn

Đỉnh tiêu đề của danh sách liên kết đơn là một đối tượng khác với một nút thông thường của danh sách. Đối tượng này lưu các thông tin:

Chỉ đến nút thực sự đầu tiên của danh sách

Chỉ đến nút cuối cùng của danh sách

Lưu giữ số lượng nút thực sự trong danh sách.

Chương trình 5.4b cài đặt lớp đỉnh tiêu đề của danh sách.

Chương trình 5.4b

```

package vidu.chuong5;

public class HeaderSimpleNode{
    private int nodeNumber;
    private SimpleNode header;
    private SimpleNode tailer;

```

```

/* Phương thức khởi dựng */
public HeaderSimpleNode(){
    nodeNumber = 0;
    header = null;
    tailer = null;
}

/* Phương thức truy nhập thuộc tính nodeNumber */
public int getNodeNumber(){
    return nodeNumber;
}
public void setNodeNumber(int nodeNumber){
    this.nodeNumber = nodeNumber;
}

/* Phương thức truy nhập thuộc tính header */
public SimpleNode getHeader(){
    return header;
}
public void setHeader(SimpleNode header){
    this.header = header;
}

/* Phương thức truy nhập thuộc tính tailer */
public SimpleNode getTailer(){
    return tailer;
}
public void setTailer(SimpleNode tailer){
    this.tailer = tailer;
}
}

```

Cài đặt danh sách liên kết đơn Danh sách liên kết đơn có thuộc tính cục bộ là một đối tượng kiểu HeaderSimpleNode. Và có các thao tác chính:

Thêm một phần tử vào một vị trí bất kì: nếu vị trí nhỏ hơn 0, thêm vào đầu danh sách. Nếu vị trí lớn hơn độ dài danh sách, thêm vào cuối. Trường hợp còn lại, chèn vào danh sách một cách bình thường.

Loại bỏ một phần tử ở vị trí bất kì: Chỉ loại bỏ khi vị trí chỉ ra nằm trong phạm vi độ dài danh sách. Phương thức này trả về nút bị loại bỏ, có kiểu SimpleNode.

Duyệt toàn bộ danh sách: Trả về giá trị của tất cả các phần tử có trong danh sách. Giá trị trả về là một mảng các phần tử giá trị có kiểu Node.

Chương trình 5.4c cài đặt lớp danh sách liên kết đơn.

Chương trình 5.4c

```
package vidu.chuong5;

public class SimpleList{
    private HeaderSimpleNode myList;

    /* Các phương thức khởi dựng */
    public SimpleList(){
        myList = new HeaderSimpleNode();
    }

    /* Phương thức chèn thêm một node vào vị trí
    @position */ public void insert(Node value, int
    position){
        // Tạo một node mới
        SimpleNode newNode = new SimpleNode(value);
        if(position <= 0){ // Chèn vào đầu
            newNode.setNext(myList.getHeader());
            myList.setHeader(newNode);
            if(myList.getNodeNumber() == 0) // Danh sách ban đầu rỗng
                myList.setTailer(newNode);
        }else if(position >= myList.getNodeNumber()){ // Chèn vào
            cuối if(myList.getNodeNumber() == 0){ // Danh sách
            ban đầu rỗng
                myList.setHeader(newNode);
                myList.setTailer(newNode);
            }else{ // Danh sách không rỗng
                myList.getTailer().setNext(newNode);
                myList.setTailer(newNode);
            }
        }
    }
}
```

```

    }else{ // Chèn vào giữa
        int index = 0;
        SimpleNode prev = null;
        SimpleNode current = myList.getHeader();
        while(index < position){
            index++;
            prev = current;
            current = current.getNext();
        }
        newNode.setNext(current);
        prev.setNext(newNode);
    }
    myList.setNodeNumber(myList.getNodeNumber()
+ 1);
}
@position */ public SimpleNode remove(int
position){
    if((myList.getNodeNumber() == 0)||
        (position < 0)|| (position >= myList.getNodeNumber()))
        return null;
    SimpleNode result = null;
    if(position == 0){ // Loại phần tử đầu
        result = myList.getHeader();
        myList.setHeader(myList.getHeader().getNext());
        if(myList.getNodeNumber() == 1) // Danh sách chỉ có 1
            phần tử
            myList.setTailer(null);
    }else if(position==myList.getNodeNumber()-1){ // Loại phần
        tử cuối result = myList.getTailer();
        SimpleNode current = myList.getHeader();
        while(!current.getNext().equals(myList.getTailer()))
            current = current.getNext();
        current.setNext(null);
        myList.setTailer(current);
    }else{ // Loại phần tử nằm giữa danh sách

```

```

        int index = 0;
        SimpleNode prev = null;
        SimpleNode current = myList.getHeader();
        while(index < position){
            index++;
            prev = current;
            current = current.getNext();
        }
        prev.setNext(current.getNext());
        result = current;
    }

    // Cập nhật số lượng node của danh sách
    myList.setNodeNumber(myList.getNodeNumber()
    - 1);

    result.setNext(null);
    return result;
}

/* Phương thức duyệt toàn bộ danh sách */
public Node[] traverse(){
    // Danh sách rỗng
    if(myList.getNodeNumber() == 0)
        return null;

    // Danh sách không rỗng
    Node[] result = new
    Node[myList.getNodeNumber()]; SimpleNode
    current = myList.getHeader(); int index = 0;
    while(current != null){
        result[index] = current.getValue();
        index++;
        current = current.getNext();
    }
    return result;
}
}

```

5.4.2 Danh sách liên kết kép

Định nghĩa một nút của danh sách liên kết kép

Một nút của danh sách liên kết kép bao gồm:

Giá trị của nút, có dạng là một đối tượng kiểu Node đã được định nghĩa trong chương trình 5.2a

Nút tiếp theo của nút đó.

Nút trước của nút đó.

Một nút của danh sách liên kết kép được cài đặt trong chương trình 5.5a.

Chương trình 5.5a

```
package vidu.chuong5;

public class DoubleNode{
    private Node value;
    private DoubleNode prev;
    private DoubleNode next;

    /* Các phương thức khởi dựng */
    public DoubleNode(){
        value = new Node();
        prev = null;
        next = null;
    }
    public DoubleNode(Node value){
        this.value = value;
        prev = null;
        next = null;
    }

    /* Phương thức truy nhập thuộc tính value */
    public Node getValue(){
        return value;
    }
    public void setValue(Node value){
        this.value = value;
    }
}
```

```

/* Phương thức truy nhập thuộc tính next */
public DoubleNode getNext(){
    return next;
}

public void setNext(DoubleNode next){
    this.next = next;
}

/* Phương thức truy nhập thuộc tính prev */
public DoubleNode getPrev(){
    return prev;
}

public void setPrev(DoubleNode prev){
    this.prev = prev;
}

```

Định nghĩa đỉnh tiêu đề của danh sách liên kết kép

Đỉnh tiêu đề của danh sách liên kết đơn là một đối tượng khác với một nút thông thường của danh sách. Đối tượng này lưu các thông tin:

Chỉ đến nút thực sự đầu tiên của danh sách

Chỉ đến nút cuối cùng của danh sách

Lưu giữ số lượng nút thực sự trong danh sách.

Chương trình 5.5b cài đặt lớp đỉnh tiêu đề của danh sách.

Chương trình 5.5b

```

package vidu.chuong5;

public class HeaderDoubleNode{
    private int nodeNumber;
    private DoubleNode header;
    private DoubleNode tailer;

    /* Phương thức khởi dựng */
    public HeaderDoubleNode(){

```

```

        nodeNumber = 0;
        header = null;
        tailer = null;
    }

    /* Phương thức truy nhập thuộc tính nodeNumber */
    public int getNodeNumber(){
        return nodeNumber;
    }
    public void setNodeNumber(int nodeNumber){
        this.nodeNumber = nodeNumber;
    }

    /* Phương thức truy nhập thuộc tính header */
    public DoubleNode getHeader(){
        return header;
    }
    public void setHeader(DoubleNode header){
        this.header = header;
    }

    /* Phương thức truy nhập thuộc tính tailer */
    public DoubleNode getTailer(){
        return tailer;
    }
    public void setTailer(DoubleNode tailer){
        this.tailer = tailer;
    }
}

```

Cài đặt danh sách liên kết kép

Danh sách liên kết kép có thuộc tính cục bộ là một đối tượng kiểu HeaderDoubleNode. Và có các thao tác chính:

Thêm một phần tử vào một vị trí bất kì: nếu vị trí nhỏ hơn 0, thêm vào đầu danh sách. Nếu vị trí lớn hơn độ dài danh sách, thêm vào cuối. Trường hợp còn lại, chèn vào danh sách một cách bình thường.

Loại bỏ một phần tử ở vị trí bất kì: Chỉ loại bỏ khi vị trí chỉ ra nằm trong phạm vi độ dài danh sách. Phương thức này trả về nút bị loại bỏ, có kiểu DoubleNode.

Duyệt toàn bộ danh sách: Trả về giá trị của tất cả các phần tử có trong danh sách. Giá trị trả về là một mảng các phần tử giá trị có kiểu Node.

Chương trình 5.5c cài đặt lớp danh sách liên kết kép.

Chương trình 5.5c

```
package vidu.chuong5;

public class DoubleList{

    private HeaderDoubleNode myList;

    /* Các phương thức khởi dựng */
    public DoubleList(){
        myList = new HeaderDoubleNode();
    }

    public void insert(Node value, int position){
        // Tạo một node mới
        DoubleNode newNode = new DoubleNode(value);
        if(position <= 0){ // Chèn vào đầu
            newNode.setNext(myList.getHeader());
            myList.getHeader().setPrev(newNode);
            myList.setHeader(newNode);
            if(myList.getNodeNumber() == 0) // Danh sách ban đầu rỗng
                myList.setTailer(newNode);
        }else if(position >= myList.getNodeNumber()){ // Chèn vào
            cuối if(myList.getNodeNumber() == 0){ // Danh sách
            ban đầu rỗng
                myList.setHeader(newNode);
                myList.setTailer(newNode);
            }else{ // Danh sách không rỗng
                newNode.setPrev(myList.getTailer());
                myList.getTailer().setNext(newNode);
                myList.setTailer(newNode);
            }
        }else{ // Chèn vào giữa
```

```

        int index = 0;
        DoubleNode current = myList.getHeader();
        while(index < position){
            index++;
            current = current.getNext();
        }
        newNode.setNext(current);
        newNode.setPrev(current.getPrev());
        current.getPrev().setNext(newNode);
        current.setPrev(newNode);
    }
    // Cập nhật số lượng node của danh sách
    myList.setNodeNumber(myList.getNodeNumber()
+ 1);
}
public DoubleNode remove(int position){
    if((myList.getNodeNumber() == 0)||
        (position < 0)|| (position >= myList.getNodeNumber()))
        return null;
    DoubleNode result = null;
    if(position == 0){ // Loại phần tử đầu
        result = myList.getHeader();
        myList.setHeader(myList.getHeader().getNext());
        if(myList.getHeader() != null)
            myList.getHeader().setPrev(null);

        if(myList.getNodeNumber() == 1) // Danh sách chỉ có 1
            phần tử myList.setTailer(null);
    }else if(position==myList.getNodeNumber()-1){ // Loại phần
        tử cuối result = myList.getTailer();
        myList.setTailer(myList.getTailer().getPrev());
        myList.getTailer().setNext(null);
    }else{ // Loại phần tử nằm giữa danh sách
        int index = 0;

```



```

        DoubleNode current = myList.getHeader();
        while(index < position){
            index++;
            current = current.getNext();
        }
        current.getPrev().setNext(current.getNext());
        current.getNext().setPrev(current.getPrev());
        result = current;
    }
    // Cập nhật số lượng node của danh sách
    myList.setNodeNumber(myList.getNodeNumber()
- 1);
    result.setPrev(null);
    result.setNext(null);
    return result;
}

/* Phương thức duyệt toàn bộ danh sách */
public Node[] traverse(){
    // Danh sách rỗng
    if(myList.getNodeNumber() == 0)
        return null;

    // Danh sách không rỗng
    Node[] result = new
Node[myList.getNodeNumber()]; DoubleNode
current = myList.getHeader(); int index = 0;
    while(current != null){
        result[index] = current.getValue();
        index++;
        current = current.getNext();
    }
    return result;
}
}

```

5.5 CÂY NHỊ PHÂN

Cài đặt nút của cây nhị phân

Một nút của cây nhị phân có các thuộc tính sau:

Giá trị của nút là một đối tượng kiểu Node

Chỉ đến nút con bên trái của nó.

Chỉ đến nút con bên phải của nó.

Chương trình 5.6a cài đặt một nút của cây nhị phân.

Chương trình 5.6a

```
package vidu.chuong5;

public class BinaryTreeNode{

    private Node value;
    private BinaryTreeNode left;
    private BinaryTreeNode right;

    /* Các phương thức khởi dựng */
    public BinaryTreeNode(){
        value = new Node();
        left = null;
        right = null;
    }

    public BinaryTreeNode(Node value){
        this.value = value;
        left = null;
        right = null;
    }

    /* Phương thức truy nhập thuộc tính value */
    public Node getValue(){
        return value;
    }

    public void setValue(Node value){
        this.value = value;
    }

    /* Phương thức truy nhập thuộc tính left */
    public BinaryTreeNode getLeft(){
```

```

        return left;
    }
    public void setLeft(BinaryTreeNode left){
        this.left = left;
    }

    /* Phương thức truy nhập thuộc tính right */
    public BinaryTreeNode getRight(){
        return right;
    }
    public void setRight(BinaryTreeNode right){
        this.right = right;
    }
}

```

Cài đặt cây nhị phân

Với cây nhị phân, ta chỉ cần lưu giữ một biến cục bộ là nút gốc của cây. Khi đó, ta cần đến các thao tác cơ bản trên cây nhị phân như sau:

 Tìm một nút có giá trị (hoặc là khoá) xác định

 Thêm nút con trái của một nút

 Thêm nút con phải của một nút

 Xoá nút con trái của một nút

 Xoá nút con phải của một nút

 Duyệt cây theo thứ tự trước

 Duyệt cây theo thứ tự giữa

 Duyệt cây theo thứ tự sau

Chương trình 5.6b cài đặt lớp cây nhị phân.

Chương trình 5.6b

```

package vidu.chuong5;
public class BinaryTree{
    private BinaryTreeNode root;

    /* Các phương thức khởi dựng */

```

```

public BinaryTree(){
    root = null;
}

public BinaryTree(Node value){
    root = new BinaryTreeNode(value);
}

/* Phương thức trả về node có giá trị @value */
public BinaryTreeNode getNode(Node value){
    return searchNode(root, value);
}

/* Phương thức tìm kiếm đệ qui một node có giá trị
   @value trên một cây con có gốc là @treeNode */
private BinaryTreeNode searchNode(BinaryTreeNode treeNode,
    Node value){
    if(treeNode.getValue().equals(value))
        return treeNode;
    if(treeNode == null)
        return null;

    BinaryTreeNode result = null;
    // Tìm trên nhánh con bên trái
    result = searchNode(treeNode.getLeft(), value);
    // Tìm trên nhánh con bên phải
    if(result == null)
        result = searchNode(treeNode.getRight(),
            value); return result;
}

/* Phương thức thêm node con bên trái của node @treeNode
   */ public boolean insertLeft(BinaryTreeNode treeNode,
    Node value){
    if((treeNode == null)|| (treeNode.getLeft() != null))
        return false;

    BinaryTreeNode newNode = new BinaryTreeNode(value);
    treeNode.setLeft(newNode);
    return true;
}

```

```

/* Phương thức thêm node con bên phải của node @treeNode */ public
boolean insertRight(BinaryTreeNode treeNode, Node value){
    if((treeNode == null)|| (treeNode.getRight() != null))
        return false;

    BinaryTreeNode newNode = new BinaryTreeNode(value);
    treeNode.setRight(newNode); return true;
}

public boolean removeLeft(BinaryTreeNode treeNode){
    // Node hiện tại rỗng
    if(treeNode == null)
        return false;

    // Node con trái không phải là node lá
    if((treeNode.getLeft() != null)&&
        ((treeNode.getLeft().getLeft() != null)||
        (treeNode.getLeft().getRight() != null)))
        return false;
    treeNode.setLeft(null);
    return true;
}

/* Phương thức xoá node con bên phải của node @treeNode */ public boolean
removeRight(BinaryTreeNode treeNode){
    // Node hiện tại rỗng
    if(treeNode == null)
        return false;

    // Node con phải không phải là node lá
    if((treeNode.getRight() != null)&&
        ((treeNode.getRight().getLeft() != null)||
        (treeNode.getRight().getRight() != null)))
        return false;
    treeNode.setRight(null);
    return true;
}

```

```

/* Phương thức duyệt cây theo thứ tự trước */
public Node[] preTraverse(){
    Node[] result = null;
    preOrder(root, result);
    return result;
}

/* Phương thức duyệt cây con @treeNode theo thứ tự trước và kết quả trả về nằm trong
   @result */
private void preOrder(BinaryTreeNode treeNode, Node[] result){ if(treeNode != null){
    addNode(result, treeNode.getValue());
    preOrder(treeNode.getLeft(), result);
    preOrder(treeNode.getRight(), result);
}
}

/* Phương thức thêm một @node vào cuối một danh sách các @nodes*/ private void
addNode(Node[] nodes, Node node){
    if(nodes == null){// Danh sách ban đầu rỗng
        nodes = new Node[1];
        nodes[0] = node;
        return;
    }
    Node[] tmpNodes = new Node[nodes.length + 1];
    for(int i=0; i<nodes.length; i++)
        tmpNodes[i] = nodes[i];
    tmpNodes[nodes.length] = node;
    nodes = tmpNodes;
}

/* Phương thức duyệt cây theo thứ tự giữa */
public Node[] inTraverse(){
    Node[] result = null;
    inOrder(root, result);
    return result;
}

```

```

}

/* Phương thức duyệt cây con @treeNode theo thứ tự giữa và kết quả trả về nằm trong
   @result */
private void inOrder(BinaryTreeNode treeNode, Node[] result){ if(treeNode != null){
    inOrder(treeNode.getLeft(), result);
    addNode(result, treeNode.getValue());
    inOrder(treeNode.getRight(), result);
}
}

/* Phương thức duyệt cây theo thứ tự sau */
public Node[] posTraverse(){
    Node[] result = null;
    posOrder(root, result);
    return result;
}

/* Phương thức duyệt cây con @treeNode theo thứ tự sau và kết quả trả về nằm trong
   @result */
private void posOrder(BinaryTreeNode treeNode, Node[] result){ if(treeNode != null){
    posOrder(treeNode.getLeft(), result);
    posOrder(treeNode.getRight(), result);
    addNode(result, treeNode.getValue());
}
}
}

```

5.6 ĐỒ THỊ

5.6.1 Biểu diễn đồ thị

Đối với đỉnh của đồ thị, để đơn giản, ta đánh số đỉnh từ 0 đến $n-1$ cho đồ thị có n đỉnh.

Đối với cạnh, ta sẽ sử dụng đồng thời hai cách biểu diễn là ma trận kề và danh sách cạnh:

Ma trận kề dùng trong các thao tác tính toán. Ma trận kề là một ma trận hai chiều $n \times n$, nếu $A[i,j]=1$ thì có cạnh từ i đến j , nếu $A[i,j]=0$ thì không có cạnh từ i đến j .

Danh sách cạnh dùng để khởi tạo đồ thị cho thuận tiện. Mỗi phần tử của danh sách là một cạnh, biểu diễn bằng hai số là hai đỉnh của đầu mút cạnh.

Chương trình 5.7a cài đặt lớp biểu diễn một cạnh của đồ thị tổng quát (có tính đến trọng số) theo danh sách cạnh.

Chương trình 5.7a

```
package vidu.chuong5;
public class Bridge{
    private int start;
    private int end;
    private int weight;

    /* Các phương thức khởi dựng */
    public Bridge(int start, int end){
        this.start = start;
        this.end = end;
        weight = 0;
    }
    public Bridge(int start, int end, int weight){
        this.start = start;
        this.end = end;
        this.weight = weight;
    }

    /* Phương thức truy nhập thuộc tính start */
    public int getStart(){
        return start;
    }
    public void setStart(int start){
        this.start = start;
    }

    /* Phương thức truy nhập thuộc tính end */
    public int getEnd(){
        return end;
    }
}
```



```

public void setEnd(int end){
    this.end = end;
}

/* Phương thức truy nhập thuộc tính weight */
public int getWeight(){
    return weight;
}

public void setWeight(int weight){
    this.weight = weight;
}
}

```

5.6.2 Cài đặt đồ thị không có trọng số

Một đồ thị không có trọng số có các thuộc tính cục bộ sau:

Số lượng các đỉnh. Từ số lượng các đỉnh có thể suy ra tập các nhãn của các đỉnh.

Ma trận kề biểu diễn các cạnh. $A[i,j]=1$ thì có cạnh từ i đến j , nếu $A[i,j]=0$ thì không có cạnh từ i đến j .

Danh sách cạnh không cần tính đến trọng số. Danh sách cạnh và ma trận kề được cập nhật đồng bộ với nhau.

Các thao tác cơ bản trên đồ thị không có trọng số:

Kiểm tra tính liên thông của đồ thị

Tìm đường đi giữa hai đỉnh bất kì

Tìm cây khung (cây bao trùm) của đồ thị Chương

trình 5.7b cài đặt lớp đồ thị không có trọng số.

Chương trình 5.7b

```

package vidu.chuong5;

public class Graph{
    private int nodeNumber; // Số lượng đỉnh
    private int[][] A;      // Ma trận kề
    private Bridge[] B;     // Danh sách cạnh

    /* Các phương thức khởi dựng */

```

```

public Graph(int nodeNumber, int[][] A){
    this.nodeNumber = nodeNumber;
    this.A = A;

    // Đồng bộ danh sách cạnh
    int lengthB = 0;
    for(int i=0; i<nodeNumber; i++)
    for(int j=0; j<nodeNumber; j++)
        if(A[i][j] == 1) lengthB ++;

    if(lengthB > 0){
        B = new Bridge[lengthB];
        int index = 0;
        for(int i=0; i<nodeNumber; i++)
        for(int j=0; j<nodeNumber; j++)
            if(A[i][j] == 1){
                B[index] = new Bridge(i,j);
                index ++;
            }
    }
}

public Graph(int nodeNumber, Bridge[] B){
    this.nodeNumber = nodeNumber;
    this.B = B;

    // Đồng bộ ma trận kề
    A = new int[nodeNumber][];
    for(int i=0; i<nodeNumber; i++)
        A[i] = new int[nodeNumber];

    for(int i=0; i<nodeNumber; i++)
    for(int j=0; j<nodeNumber; j++)
        A[i][j] = 0;

    if(B != null){
        for(int i=0; i<B.length; i++){
            A[B[i].getStart()][B[i].getEnd()] = 1;

```

```

    }
}

/* Phương thức truy nhập thuộc tính nodeNumber */
public int getNodeNumber(){
    return nodeNumber;
}

public void setNodeNumber(int nodeNumber){
    this.nodeNumber = nodeNumber;
}

/* Thêm một cạnh vào đồ thị */
public void addBridge(Bridge bridge){
    if(B == null){ // Ban đầu chưa có cạnh nào B = new Bridge[1];

        B[0] = bridge;
    }else{
        Bridge *tmp = new Bridge[B.length + 1];
        for(int i=0; i<B.length; i++)
            tmp[i] = B[i];
        tmp[B.length] = bridge;
    }
    // Cập nhật ma trận kề
    A[bridge.getStart()][bridge.getEnd()] = 1;
}

/* Kiểm tra tính liên thông của đồ thị */
public boolean isConnected(){
    // Mảng đánh dấu duyệt node
    boolean[] visited = new boolean[nodeNumber];
    for(int i=0; i<nodeNumber; i++)
        visited[i] = false;
    int[] queue = new int[nodeNumber]; // Hàng đợi, duyệt BFS
    int front=0, tail = 0;
    int size = 1; // Số lượng đỉnh liên thông
    queue[0] = 0;

```

```

visited[0] = true; // Khởi tạo hàng đợi
while(front <= tail){ // Đếm số đỉnh liên thông
    int u = queue[front];
    front++;
    for(int j=0; j<nodeNumber; j++){
        if(A[u][j] == 1)&&(!visited[j]){
            tail++;
            queue[tail] = j;
            visited[j] = true;
            size++;
        }
    }
    // Nếu số đỉnh liên thông bằng nodeNumber thì đồ thị là liên
    thông if(size == nodeNumber) return true; return false;
}

public Bridge[] way(int start, int end){
    // Mảng đánh dấu duyệt node
    boolean[] visited = new boolean[nodeNumber];
    for(int i=0; i<nodeNumber; i++){
        visited[i] = false;
    }
    int[] prev = new int[nodeNumber]; // Mảng lưu vết đường đi
    int[] queue = new int[nodeNumber]; // Hàng đợi, duyệt BFS
    int front=0, tail = 0;
    queue[0] = start;
    visited[start] = true; // Khởi tạo hàng đợi
    while(front <=
    tail)&&(!visited[end]){ // Tìm đường đi
        int u = queue[front];
        front++;
        for(int j=0; j<nodeNumber; j++){
            if(A[u][j] ==
            1)&&(!visited[j]){
                tail++;
                queue[tail] = j;
                visited[j] = true;
                prev[j] = u;
            }
        }
    }
}

```

```

    }

}

// nếu chưa đến được node @end thì không có
đường đi if(!visited[end]) return null;

/* Trường hợp có đường đi, Lưu vết vào một
stack */ int[] stack = new int[nodeNumber]; int
top = 0;
stack[top] = end;
while(stack[top] != start){
    int v = prev[stack[top]];
    top++;
    stack[top] = v;
}
/* Đọc kết quả từ stack */
Bridge[] result = new Bridge[top];
int index = 0;
while(top > 0){
    result[index] = new Bridge(stack[top], stack[top-1]);
    index++;
    top--;
}
return result;
}

public Graph tree(){
    Nếu đồ thị không liên thông, sẽ không có cây bao
    trùm if(!isConnected())
        return null;

    Khởi tạo cây bao trùm, cũng là một đồ thị có @nodeNumber node
    int[][] newA = new int[nodeNumber][];
    for(int i=0; i<nodeNumber; i++)
        newA[i] = new int[nodeNumber];
    for(int i=0; i<nodeNumber; i++)
    for(int j=0; j<nodeNumber; j++)
        newA[i][j] = 0;
    // Chưa có cạnh nào

```

```

Graph result = new Graph(nodeNumber, newA);

// Mảng đánh dấu duyệt node
boolean[] visited = new boolean[nodeNumber];
for(int i=0; i<nodeNumber; i++)
    visited[i] = false;
int[] queue = new int[nodeNumber]; // Hàng đợi, duyệt BFS
int front=0, tail = 0;
queue[0] = 0;
visited[0] = true; // Khởi tạo hàng đợi
while(front <= tail){ // Tìm cạnh của CBT
    int u = queue[front];
    front++;
    for(int j=0; j<nodeNumber; j++)
        if(A[u][j] == 1 && (!visited[j])){
            tail++;
            queue[tail] = j;
            visited[j] = true;
            result.addBridge(new Bridge(u,j)); // Bổ sung cạnh vào CBT
        }
    }
return result;
}
}

```

5.6.3 Cài đặt đồ thị có trọng số

Một đồ thị có trọng số có các thuộc tính cục bộ sau:

Số lượng các đỉnh. Từ số lượng các đỉnh có thể suy ra tập các nhãn của các đỉnh.

Ma trận kề biểu diễn các cạnh có tính đến trọng số: $A[i,j]$ = trọng số cạnh ij . Nếu giữa i và j không có cạnh thì giá trị này là vô cùng (tự định nghĩa trong chương trình - `maxWeight`)

Danh sách cạnh có tính đến thuộc tính trọng số (trọng số có thể âm, nhưng giả sử không có chu trình âm). Danh sách cạnh và ma trận kề được cập nhật đồng bộ với nhau.

Các thao tác cơ bản trên đồ thị có trọng số:

Kiểm tra tính liên thông của đồ thị

Tìm đường đi ngắn nhất giữa hai đỉnh bất kì

Tìm cây khung (cây bao trùm) nhỏ nhất của đồ thị

Chương trình 5.7c cài đặt lớp đồ thị có trọng số.

Chương trình 5.7c

```
package vidu.chuong5;

public class WeightedGraph{

    private int nodeNumber; // Số lượng đỉnh
    private int[][] A;      // Ma trận kề
    private Bridge[][] B;   // Danh sách cạnh
    private static int maxWeight = 10000;

    /* Các phương thức khởi dựng */
    public WeightedGraph(int nodeNumber, int[][] A){
        this.nodeNumber = nodeNumber;
        this.A = A;

        // Đồng bộ danh sách cạnh
        int lengthB = 0;
        for(int i=0; i<nodeNumber; i++)
            for(int j=0; j<nodeNumber; j++)
                if(A[i][j] < maxWeight) lengthB ++;

        if(lengthB > 0){
            B = new Bridge[lengthB];
            int index = 0;
            for(int i=0; i<nodeNumber; i++)
                for(int j=0; j<nodeNumber; j++)
                    if(A[i][j] < maxWeight){
                        B[index] = new Bridge(i,j, A[i][j]);
                        index ++;
                    }
        }
    }

    public WeightedGraph(int nodeNumber,
        Bridge[] B){ this.nodeNumber =
        nodeNumber;
```

```

this.B = B;

// Đồng bộ ma trận kề
A = new int[nodeNumber][];
for(int i=0; i<nodeNumber; i++)
    A[i] = new int[nodeNumber];

for(int i=0; i<nodeNumber; i++)
for(int j=0; j<nodeNumber; j++)
    A[i][j] = maxWeight;

if(B != null){
    for(int i=0; i<B.length; i++){
        A[B[i].getStart()][B[i].getEnd()] =
            B[i].getWeight();
    }
}

}

/* Phương thức truy nhập thuộc tính nodeNumber */
public int getNodeNumber(){
    return nodeNumber;
}

public void setNodeNumber(int nodeNumber){
    this.nodeNumber = nodeNumber;
}

/* Thêm một cạnh vào đồ thị */
public void addBridge(Bridge bridge){
    if(B == null){ // Ban đầu chưa có cạnh nào B = new
        Bridge[1];

        B[0] = bridge;
    }else{
        Bridge[] tmp = new Bridge[B.length + 1];
        for(int i=0; i<B.length; i++)
            tmp[i] = B[i];
        tmp[B.length] = bridge;
    }
}

```



```

// Cập nhật ma trận kề
A[bridge.getStart()][bridge.getEnd()] = bridge.getWeight();
}

public boolean isConnected(){
    // Mảng đánh dấu duyệt node
    boolean[] visited = new boolean[nodeNumber];
    for(int i=0; i<nodeNumber; i++)
        visited[i] = false;
    int[] queue = new int[nodeNumber]; // Hàng đợi, duyệt BFS
    int front=0, tail = 0;
    int size = 1; // Số lượng đỉnh liên thông
    queue[0] = 0;
    visited[0] = true; // Khởi tạo hàng đợi
    while(front <= tail){ // Đếm số đỉnh liên thông
        int u = queue[front];
        front++;
        for(int j=0; j<nodeNumber; j++){
            if(A[u][j] < maxWeight)&&(!visited[j]){
                tail++;
                queue[tail] = j;
                visited[j] = true;
                size++;
            }
        }
    }

    Nếu số đỉnh liên thông bằng nodeNumber thì đồ thị là liên thông
    if(size == nodeNumber) return true;
    return false;
}

/* Tìm đường đi ngắn nhất giữa hai đỉnh bất kì, Ford-
Bellman */ public Bridge[] minWay(int start, int end){
    int[] prev = new int[nodeNumber]; // Mảng lưu vết đường đi
    int[] distance = new
int[nodeNumber]; // Khoảng cách đến các node
    for(int i=0; i<nodeNumber; i++){
        prev[i] = start;

```

```

        distance[i] = maxWeight;
    }
    distance[start] = 0;
    for(int i=0; i<nodeNumber-2; i++)
    for(int j=0; j<nodeNumber; j++)
    for(int k=0; k<nodeNumber; k++)
        if(j != start)&&(distance[j] > distance[k] + A[k][j]){
            distance[j] = distance[k] + A[k][j];
            prev[j] = k;
        }
    }
    // nếu chưa đến được node @end thì không có đường đi if(distance[end] ==
    maxWeight) return null;
    /* Trường hợp có đường đi, Lưu vết vào một stack */ int[] stack = new
    int[nodeNumber]; int top = 0;
    stack[top] = end;
    while(stack[top] != start){
        int v = prev[stack[top]];
        top++;
        stack[top] = v;
    }
    /* Đọc kết quả từ stack */
    Bridge[] result = new Bridge[top];
    int index = 0;
    while(top > 0){
        result[index] = new Bridge(stack[top], stack[top-1]);
        index++; top--;
    }
    return result;
}

/* Tìm cây khung nhỏ nhất của đồ thị */
public WeightedGraph minTree(){
    // Nếu đồ thị không liên thông, sẽ không có cây bao trùm if(!isConnected())
    return null;
}

```

```

// Khởi tạo cây bao trùm, cũng là một đồ thị có @nodeNumber node
int[][] newA =
new int[nodeNumber][]; for(int i=0; i<nodeNumber; i++)
    newA[i] = new int[nodeNumber];
for(int i=0; i<nodeNumber; i++)
for(int j=0; j<nodeNumber; j++)
    newA[i][j] = maxWeight; // Chưa có cạnh nào
WeightedGraph result = new
WeightedGraph(nodeNumber, newA);
// Sắp xếp cạnh không giảm
for(int i=0; i<B.length; i++)
for(int j=i+1; j<B.length; j++)
    if(B[i].getWeight() > B[j].getWeight()){
        Bridge tmp = B[i];
        B[i] = B[j];
        B[j] = tmp;
    }
int index = 0, bridgeNumber = 0;
while(bridgeNumber < nodeNumber)&&(index < B.length){
    if(result.minWay(B[index].getStart(),B[index].getEnd())==null){
        result.addBridge(
            new Bridge(B[index].getStart(),B[index].getEnd()));
        bridgeNumber++;
    }
    index++;
}
return result;
}
}

```

5.7 CASE STUDY III

Trong phần này, ta sẽ sử dụng các lớp đối tượng đã được cài đặt trong chương này: ngăn xếp, hàng đợi, danh sách liên kết, cây nhị phân và đồ thị.

Chương trình 5.8 cài đặt một chương trình tổng hợp, sử dụng menu cho các đối tượng: stack, queue, list. Với mỗi đối tượng, có một menu con tương ứng với các thao tác trên đối tượng đó. Với các đối tượng còn lại, được coi như bài tập mở rộng của phần này.

Chương trình 5.8

```
package vidu.chuong5;

public class CaseStudy3{

    public static void main(String[] args) {

        try {

            BufferedReader br =

                new BufferedReader(new InputStreamReader(System.in));

            while (true) {

                System.out.println("*****");

                System.out.println("      CASE STUDY 3          ");

                System.out.println(" -----oOo-----      ");

                System.out.println("1.      Thao tac voi ngan xep          ");

                System.out.println("2.      Thao tac voi hang doi          ");

                System.out.println("Thao tac voi danh          ");

                System.out.println("3.      sach                          ");

                System.out.println("10.Thoat!                          ");

                System.out.println("*****");

                System.out.print(" |==> Chon chuc nang: ");

                int function = Integer.parseInt(br.readLine());

                CaseStudy3 ob3 = new CaseStudy3();

                switch(function){

                    case 1: // Thao tác với ngăn xếp

                        ob3.stack(br);

                        break;

                    case 2: // Thao tác với hàng đợi

                        ob3.queue(br);

                        break;

                    case 3: // Thao tác với danh sách

                        ob3.list(br);

                        break;

                    case 10: // Thoát khỏi chương trình

                        System.exit(1);

                }

            }

        }

    }

}
```

```

    }
    catch(Exception ex)
    {
        System.out.println("Loi o ham 'main': ");
        ex.printStackTrace();
    }
}
/* Thao tác với ngăn xếp */
public void stack(BufferedReader br){
    MyStack stack = new MyStack();
    try {
        while (true) {
            System.out.println("*****");
            THAO TAC VOI NGAN
            System.out.println(" XEP ");
            System.out.println(" -----oOo----- ");
            System.out.println("
1. Them mot phan tu vao stack ");
            System.out.println(" Lay mot phan tu ra khoi stack
2. ");
            System.out.println(" Lay het cac phan tu cua stack
3. ");
            System.out.println(" 10.Thoat! ");
            System.out.println("*****");
            System.out.print(" |==> Chon chuc nang: ");
            int function = Integer.parseInt(br.readLine());
            switch(function){
                case 1: // Thêm một phần tử vào ngăn xếp
                    System.out.print("Gia tri phan tu them vao:");
                    int phantu = Integer.parseInt(br.readLine());
                    stack.push(new Node(phantu));
                    break;
                case 2: // Lấy một phần tử ra khỏi ngăn xếp
                    Node node = stack.pop();
                    if(node != null)
                        System.out.println("Phan tu lay ra: "
                            + node.getValue());
                    else
                        System.out.println("Stack da rong!");
            }
        }
    }
}

```

```

        break;
    case 3: // Lấy tất cả các phần tử của ngăn xếp System.out.println("Cac
        phan tu cua ngan xep:"); while(true){
        Node node = stack.pop();
        if(node == null)
            break;
        System.out.print(" " + node.getValue());
        }
        break;
    case 10: // Thoát khỏi chương trình con
        return;
    }
}
}
catch(Exception ex)
{
    ex.printStackTrace();
}
}

public void queue(BufferedReader br){
    MyQueue queue = new MyQueue();
    try {
        while (true) {
            System.out.println("*****");

            System.out.println    THAO TAC VOI HANG
            ("                    DOI                    ");
            System.out.println(" -----oOo----- ");
            System.out.println
            ("                1. Them mot phan tu vao queue ");
            System.out.println
            ("                2. Lay mot phan tu ra khoi queue ");
            System.out.println    Lay het cac phan tu cua queue
            ("                3. ");
            System.out.println(" 10.Thoat! ");
            System.out.println("*****
            ");
            |==> Chon chuc nang:
            System.out.print(" ");

```

```
int function = Integer.parseInt(br.readLine());
```

```

switch(function){
    case 1: // Thêm một phần tử vào hàng đợi
        System.out.print("Gia tri phan tu them vao:");
        int phantu = Integer.parseInt(br.readLine());
        queue.insert(new Node(phantu));
        break;
    case 2: // Lấy một phần tử ra khỏi hàng đợi
        Node node = queue.remove();
        if(node != null)
            System.out.println("Phan tu lay ra: "
                + node.getValue());
        else
            System.out.println("Khong co phan tu nao!");
        break;
    case 3: // Lấy tất cả các phần tử của hàng đợi
        System.out.println("Cac phan tu cua hang
            doi:"); while(true){
            Node node = queue.remove();
            if(node == null)
                break;
            System.out.print(" " + node.getValue());
        }
        break;
    case 10: // Thoát khỏi chương trình con
        return;
    }
}
}
}
catch(Exception ex)
{
    ex.printStackTrace();
}
}
/* Thao tác với danh sách liên kết đơn */
public void list(BufferedReader br){
    SimpleList list = new SimpleList();

```

```

try {
    while (true) {
        System.out.println("*****");
        System.out.println("  THAO TAC VOI DANH SACH DON  ");
        System.out.println(" -----o0o----- ");
        System.out.println(" 1. Them mot phan tu vao list      ");
        System.out.println(" 2. Lay mot phan tu ra khoi list    ");
        System.out.println(" 3. Duyet cac phan tu cua list     ");
        System.out.println(" 10.Thoat!                          ");
        System.out.println("*****");
        ***"); System.out.print(" |==> Chon chuc nang: ");
        int function = Integer.parseInt(br.readLine());
        switch(function){
            case 1: // Thêm một phần tử vào danh sách
                System.out.print("Gia tri phan tu them vao:");
                int phantu = Integer.parseInt(br.readLine());
                System.out.print("Vi tri chen phan tu:");
                int vitri = Integer.parseInt(br.readLine());
                list.insert(new Node(phantu), vitri);
                break;
            case 2: // Lấy một phần tử ra khỏi danh sách
                System.out.print("Vi tri phan tu lay ra:");
                int vitri = Integer.parseInt(br.readLine());
                SimpleNode node = list.remove(vitri);
                if(node != null)
                    System.out.println("Phan tu lay ra: "
                        + node.getValue());
                else
                    System.out.println("Khong co phan tu nao!");
                break;
            case 3: // Lấy tất cả các phần tử của danh sách
                System.out.println("Cac phan tu cua danh
sach:"); Node[] nodes = list.traverse();
                for(int i=0; i<nodes.length; i++)
                    System.out.print(" " + nodes[i].getValue());
                break;

```



```

        case 10: // Thoát khỏi chương trình con
            return;
        }
    }
}
catch(Exception ex)
{
    ex.printStackTrace();
}
}
}

```

TỔNG KẾT CHƯƠNG 5

Nội dung chương 5 đã trình bày việc biểu diễn và cài đặt một số cấu trúc dữ liệu trừu tượng trên Java, bao gồm các giải thuật:

Phương pháp duyệt

Phương pháp đệ qui

Phương pháp sắp xếp

Phương pháp tìm kiếm.

Và các cấu trúc dữ liệu trừu tượng:

Ngăn xếp (stack)

Hàng đợi (queue)

Danh sách liên kết (list): danh sách liên kết đơn và danh sách liên kết kép.

Cây nhị phân (binary tree)

Đồ thị (graph): đồ thị không trọng số và đồ thị có trọng số.

Hơn nữa, chương này cũng đã trình bày và minh họa cách sử dụng các đối tượng trừu tượng trong các ứng dụng.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5

Cài đặt bổ sung các phương thức sắp xếp khác vào lớp Array trong mục 5.2.1:

Sắp xếp nổi bọt

Sắp xếp chọn.

Sắp xếp vun đống

Sắp xếp trộn

Chuyển lớp MyStack trong mục 5.3.1 thành một ngăn xếp của các kí tự. Sau đó, dùng lớp ngăn xếp mới đó để đảo ngược một chuỗi kí tự do người dùng nhập vào từ bàn phím.

Dùng lớp MyQueue để mô phỏng chương trình quản lí tiến trình đơn giản của hệ điều hành:

- Mỗi tiến trình được mã hoá bằng một kí hiệu có kiểu int.

- Khi một tiến trình xuất hiện, nó sẽ được đưa vào một hàng đợi.

- Khi CPU rỗi, một tiến trình trong hàng đợi sẽ được lấy ra thực hiện.

Yêu cầu viết menu thực hiện chương trình:

- Thêm một tiến trình

- Lấy một tiến trình ra

- Xem tất cả các tiến trình đang có mặt trong hàng đợi.

Thay đổi cấu trúc lớp Node của danh sách liên kết, sao cho value có kiểu là lớp Employee đã được định nghĩa trong chương 4.

Dùng danh sách liên kết đơn với cấu trúc mới trong bài 4 để quản lí nhân viên của một công ty: Thêm vào một nhân viên, xoá đi một nhân viên, tìm kiếm một nhân viên theo một tiêu chí cụ thể (tiền lương, tuổi...), duyệt tất cả các nhân viên.

Dùng danh sách liên kết kép để thực hiện bài số 5.

Viết chương trình (dạng menu) để áp dụng thực hiện các thao tác trên cây nhị phân, sử dụng lớp BinaryTree trong mục 5.5.

Viết chương trình (dạng menu) để áp dụng thực hiện các thao tác trên đồ thị không trọng số, sử dụng lớp Graph trong mục 5.6.1.

Viết chương trình (dạng menu) để áp dụng thực hiện các thao tác trên đồ thị có trọng số, sử dụng lớp WeightedGraph trong mục 5.6.2.

CHƯƠNG 6

LẬP TRÌNH GIAO DIỆN TRÊN JAVA

Nội dung chương này tập trung trình bày các vấn đề liên quan đến lập trình giao diện, với sự hỗ trợ của một số đối tượng được cung cấp sẵn bởi Java:

Lập trình giao diện với các đối tượng cơ bản và với các đối tượng multimedia.

Lập trình giao diện với HTML&Applet

Lập trình giao diện với SWING

6.1 GIAO DIỆN VỚI CÁC ĐỐI TƯỢNG CƠ BẢN

Trong mục này, chúng ta sẽ tìm hiểu và sử dụng các đối tượng cơ bản của lập trình giao diện trong Java:

Các đối tượng khung chứa (container) cơ bản: Frame, Panel, Dialog.

Các đối tượng thành phần (component) cơ bản: Button, Label, TextField, TextArea

Các sự kiện cơ bản của các đối tượng.

Muốn sử dụng các đối tượng này, cần thêm lệnh sử dụng thư viện awt của Java:

```
import java.awt.*;
```

6.1.1 Các đối tượng container cơ bản

Các đối tượng container được dùng để chứa các đối tượng thành phần khác. Các lớp đối tượng này có một số phương thức chung như sau:

add(Object): Thêm một đối tượng (kiểu component) vào container.

remove(Object): Loại bỏ một đối tượng ra khỏi container.

removeAll(): Loại bỏ tất cả các đối tượng mà container đang chứa.

getComponent(int): Trả về đối tượng thành phần có chỉ số là tham số đầu vào.

Container quản lý các đối tượng chứa trong nó dưới dạng mảng.

Chỉ số của các thành phần là số thứ tự khi thành phần đó được thêm vào container.

getComponents(): Trả về mảng tất cả các đối tượng mà container đang chứa.

countComponents(): Trả về số lượng các đối tượng mà container đang chứa.

Frame

Frame là một đối tượng có thể dùng một cách độc lập, hoặc được gắn vào một đối tượng khác như một đối tượng component bình thường. Thông thường, Frame được dùng như một cửa sổ của một chương trình độc lập. Các phương thức cơ bản của lớp Frame:

Frame(): Khởi tạo không tham số.

Frame(String): Khởi tạo với tham số là dòng tiêu đề của frame.

setSize(int, int): Định kích cỡ của frame, tham số tương ứng là chiều rộng và chiều cao của frame.

setVisible(boolean): Cho phép frame xuất hiện hay ẩn đi trên màn hình.

setTitle(String)/getTitle(): Truy nhập thuộc tính dòng tiêu đề của frame.

setResizable(boolean): Thiết lập thuộc tính cho phép thay đổi kích cỡ frame.

setIconImage(Image): Thiết lập ảnh icon ở góc trên (biểu tượng) của frame.

Chương trình 6.1 minh họa việc sử dụng một đối tượng của lớp Frame.

Chương trình 6.1

```
package vidu.chuong6;
import java.awt.*;
public class FrameDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("This is my Frame!");
        myFrame.setSize(300,150); // Định kích cỡ frame
        myFrame.setVisible(true);    // Hiển thị frame
    }
}
```



Hình 6.1: Kết quả demo Frame

Panel

Panel cũng là một dạng khung chứa, nhưng khá đơn giản. Panel chỉ dùng để nhóm các đối tượng giao diện với nhau. Thông thường, panel được dùng trong một cửa sổ của Frame hoặc một ứng dụng khác. Các phương thức cơ bản của lớp Panel, ngoài các phương thức chung của container:

Panel(): Khởi tạo không tham số.

Chương trình 6.2 minh họa việc sử dụng một Panel trong một Frame.

Chương trình 6.2

```

package vidu.chuong6;
import java.awt.*;
public class PanelDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has a
        panel!"); myFrame.setSize(300,150); // Định
        kích cỡ frame Panel myPanel = new Panel();//
        Khai báo panel
        myFrame.add(myPanel);           // Thêm panel vào frame
        myFrame.setVisible(true);       // Hiển thị frame
    }
}

```



Hình 6.2: Kết quả demo Panel

Dialog

Dialog là một đối tượng cửa sổ con của một cửa sổ chương trình chính. Do vậy, Dialog chỉ được sử dụng kèm với một Frame. Có hai dạng Dialog:

Modal: Khi hiện cửa sổ dialog, các cửa sổ khác của chương trình sẽ bị khoá lại, không thao tác được, chỉ thao tác được trên cửa sổ dialog.

Non-modal: Không khoá các cửa sổ khác. Khi dialog xuất hiện, người dùng vẫn có thể chuyển sang thao tác trên các cửa sổ khác, nếu cần.

Các phương thức cơ bản của lớp Dialog:

Dialog(Frame, boolean): Khởi tạo dialog, tham số thứ nhất là frame chứa dialog, tham số thứ hai xác định dialog có là modal hay không.

Dialog(Frame, String, boolean): Khởi tạo dialog, thêm tham số thứ hai là dòng tiêu đề của dialog.

setVisible(boolean): Thiết lập trạng thái hiển thị hoặc ẩn dialog trên màn hình.

setSize(int, int): Định kích cỡ cho dialog, các tham số tương ứng là chiều rộng và chiều cao của dialog.

setTitle(String)/getTitle(): Truy nhập thuộc tính dòng tiêu đề của dialog.

setResizable(boolean): Thiết lập thuộc tính cho phép thay đổi kích cỡ của dialog.

setLayout(Layout): Thiết lập chế độ hiển thị các đối tượng chứa trong dialog.

Chương trình 6.3 minh họa việc thêm một dialog (đang rỗng, chưa có đối tượng thành phần nào) vào một frame.

Chương trình 6.3

```
package vidu.chuong6;
import java.awt.*;
public class DialogDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has a
        dialog!"); myFrame.setSize(300,150); // Định
        kích cỡ frame // Khai báo và khởi tạo dialog
        Dialog myDialog = new Dialog(myFrame, "An empty dialog!",
        true); myDialog.setSize(300,150); // Định kích cỡ dialog
        myDialog.setVisible(true); // Hiển thị dialog
    }
}
```



Hình 6.3: Kết quả demo Dialog

6.1.2 Các đối tượng component cơ bản

Các đối tượng component được dùng để làm thành phần của các đối tượng khung chứa, chúng không thể dùng độc lập, mà luôn phải gắn vào trong một đối tượng khung chứa container.

Label

Label (nhãn) là một đối tượng để hiển thị văn bản tĩnh, những văn bản mà người dùng không thể thay đổi trực tiếp được. Các phương thức cơ bản của Label:

Label(): Khởi tạo một nhãn rỗng.

Label(String): Khởi tạo một nhãn với nội dung văn bản là tham số đầu vào.

Label(String, int): Khởi tạo một nhãn có nội dung sẵn, tham số thứ hai xác định cách căn lề của nhãn so với khung chứa, bao gồm {Label.CENTER, Label.LEFT, Label.RIGHT}.

setText(String)/getText(): Truy nhập nội dung văn bản của nhãn.

setAlignment(int)/getAlignment(): Truy nhập thuộc tính căn lề của nhãn.

setFont(Font): Định dạng phông chữ của nhãn.

Chương trình 6.4 minh họa việc sử dụng nhãn trong một frame.

Chương trình 6.4

```
package vidu.chuong6;
import java.awt.*;
public class LabelDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has a label!");
        myFrame.setSize(300,150);           // Định kích cỡ frame

        // Khai báo và khởi tạo label
        Label myLabel = new Label();
        myLabel.setText("This is a label!"); // Gán nội dung
        // văn bản myLabel.setAlignment(Label.CENTER); //
        // Căn lề giữa
        myFrame.add(myLabel);               // Gắn label vào frame
        myFrame.setVisible(true);           // Hiển thị frame
    }
}
```



Hình 6.4: Kết quả demo Label

TextField và TextArea

Đây là hai đối tượng dùng để biểu diễn văn bản và người dùng có thể thay đổi nội dung văn bản chứa trong chúng. Điểm khác biệt là TextField chỉ cho phép một dòng văn bản, trong khi TextArea cho phép chứa nhiều dòng văn bản. Các phương thức chung của hai lớp này:

`setText(String)/getText()`: Truy nhập thuộc tính nội dung văn bản chứa trong ô.

`getSelectedText()`: Trả về chuỗi văn bản được bôi đen (đánh dấu chọn) trong ô.

`getSelectedStart()`: Trả về vị trí kí tự đầu trong vùng được đánh dấu chọn (tính từ 0).

`getSelectedEnd()`: Trả về vị trí kí tự cuối trong vùng được đánh dấu chọn (tính từ 0).

`selectAll()`: Đánh dấu chọn toàn văn bản.

`setEditable(boolean)`: Xác định vùng văn bản có thể edit được hay không.

Các phương thức khác của lớp `TextField`:

`TextField()`: Khởi tạo một ô văn bản rỗng.

`TextField(int)`: Khởi tạo một ô văn bản rỗng, độ rộng xác định bởi tham số vào.

`TextField(String)`: Khởi tạo một ô văn bản có nội dung xác định bởi tham số đầu vào.

`TextField(String, int)`: Khởi tạo với nội dung có sẵn, độ rộng xác định.

`setEchoChar(char)/getEchoChar()`: Truy nhập thuộc tính là kí tự thay thế văn bản trong ô. Thuộc tính này được dùng khi ta cần che dấu thông tin văn bản, ví dụ, ô gõ mật khẩu của chương trình.

`getColumns()`: Trả về độ rộng của ô văn bản.

Các phương thức khác của lớp `TextArea`:

`TextArea()`: Khởi tạo một vùng văn bản rỗng.

`TextArea(int, int)`: Khởi tạo một vùng văn bản rỗng, kích cỡ (số dòng, số cột) xác định bởi tham số vào.

`TextArea(String)`: Khởi tạo một vùng văn bản có nội dung xác định bởi tham số đầu vào.

`TextArea(String, int, int)`: Khởi tạo vùng văn bản với nội dung có sẵn, độ rộng xác định.

`appendText(String)`: Thêm một đoạn văn bản vào cuối đoạn văn bản trong vùng.

`insertText(String, int)`: Chèn một đoạn văn bản vào vị trí xác định (tham số thứ hai) của vùng văn bản.

`replaceText(String, int, int)`: Thay thế một đoạn văn bản trong vùng, đánh dấu bằng vị trí bắt đầu và vị trí kết thúc (tham số thứ hai và thứ ba), bằng một đoạn văn bản mới (tham số thứ nhất).

getRows()/getColumns(): Trả về số dòng/cột của vùng văn bản.

Chương trình 6.5 minh họa việc đặt các đối tượng ô văn bản và vùng văn bản vào một frame.

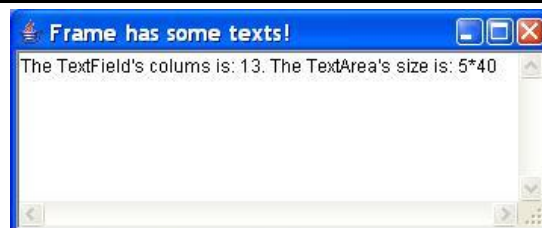
Chương trình 6.5

```
package vidu.chuong6;
import java.awt.*;
public class TextDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has some texts!");
        myFrame.setSize(300,150);           // Định kích cỡ frame

        // Khai báo và khởi tạo textField
        TextField myTextField = new TextField("A text field!");
        myFrame.add(myTextField);           // Gắn vào frame

        // Khai báo và khởi tạo textArea
        TextArea myTextArea = new TextArea(5, 40);
        String str="The TextField's columns is: "+myTextField.getColumns();
        str += "The TextArea's size is: " + myTextArea.getRows()
            + "*" + myTextArea.getColumns();
        myTextArea.setText(str);             // Thiết lập nội dung
        myFrame.add(myTextArea);            // Gắn vào frame

        myFrame.setVisible(true);          // Hiển thị frame
    }
}
```



Hình 6.5: Kết quả demo Text

Button

Button là đối tượng nút lệnh, dùng để thực hiện một nhiệm vụ xác định. Các phương thức cơ bản của nút nhấn:

Button(String): Khởi tạo nút nhấn với tên xác định trên nút.

setLabel(String)/getLabel(): Truy nhập tên của nút nhấn.

Chương trình 6.6 minh họa việc tạo một nút nhấn trong một frame.

Chương trình 6.6

```
package vidu.chuong6;
import java.awt.*;
public class ButtonDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has a button!");
        myFrame.setSize(300,150);           // Định kích cỡ frame

        // Khai báo và khởi tạo button
        Button myButton = new Button("Click!");
        myFrame.add(myButton);              // Gắn vào frame
        myFrame.setVisible(true);          // Hiển thị frame
    }
}
```



Hình 6.6: Kết quả demo Button

Tuy nhiên, khi click vào nút nhấn này, không xảy ra điều gì cả. Lí do là chúng ta chưa cài đặt việc xử lí sự kiện cho nút nhấn. Nội dung phần 6.1.3 sẽ trình bày việc xử lí sự kiện cho các đối tượng.

6.1.3 Các sự kiện cơ bản của đối tượng

Mỗi đối tượng component có một số sự kiện xác định, phát sinh từ chính đối tượng đó.

Java cung cấp một số lớp sự kiện cơ bản nằm trong thư gói java.awt.event:

```
import java.awt.event.*;
```

Các lớp sự kiện cơ bản của các đối tượng bao gồm:

ActionEvent: Xuất hiện khi một nút bị click vào, một danh sách (list) được chọn, một menu được chọn.

ComponentEvent: Xuất hiện khi một component bị thay đổi kích cỡ, vị trí, trạng thái.

FocusEvent: Xuất hiện khi một component có hoặc mất focus.

ItemEvent: Xuất hiện khi một menu item được chọn hoặc bỏ, khi checkbox hoặc list item được click vào.

WindowEvent: Xuất hiện khi một cửa sổ được mở ra, kích hoạt, đóng lại hoặc thoát ra.

TextEvent: Xuất hiện khi giá trị văn bản của các đối tượng TextField và TextArea bị thay đổi.

MouseEvent: Xuất hiện khi chuột được click, di chuyển qua, nhấn xuống và thả ra.

KeyEvent: Xuất hiện khi có đầu vào từ bàn phím.

Các giao tiếp được cài đặt để xử lý các sự kiện trên:

ActionListener.

ComponentListener

FocusListener

ItemListener

WindowListener

TextListener

MouseListener và MouseMotionListener

KeyListener

Khi cài đặt các giao tiếp này, cần cài đặt lại phương thức xử lý sự kiện:

```
public void actionPerformed(<Đối tượng lớp sự kiện>){  
    ... // Cài đặt lại mã lệnh  
}
```

Để xác định sự kiện phát sinh từ component nào, ta dùng phương thức getSource():

```
<Kiểu component> <Đối tượng sự kiện>.getSource();
```

Chương trình 6.7 cài đặt một ứng dụng hoàn chỉnh, bao gồm:

Hai nhãn tiêu đề cho hai ô văn bản.

Hai ô văn bản, để nhập số liệu vào.

Bốn nút nhấn tương ứng để thực hiện các thao tác nhân, chia, cộng, trừ các số liệu nhập từ hai ô văn bản.

Thêm một nút nhấn, khi click vào sẽ thoát khỏi chương trình (chương trình kết thúc).

Chương trình 6.7

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
public class EventDemo extends Frame implements ActionListener{ Label lbl1, lbl2, lblKq;
    TextField txt1, txt2;
    Button btnCong, btnTru, btnNhan, btnChia, btnThoat;
    public EventDemo(){
        super("Event demo!");
        this.setLayout(new GridLayout(6,2));    //Chế độ hiển thị 6 dòng, 2 cột
        lbl1 = new Label("So thu
nhat:");                                     // Nhãn số thứ nhất
        this.add(lbl1);
        txt1 = new TextField();                // Ô văn bản số thứ nhất
        this.add(txt1);
        lbl2 = new Label("So thu hai:"); // Nhãn số thứ hai
        this.add(lbl2);
        txt2 = new TextField();                // Ô văn bản số thứ hai
        this.add(txt2);
        lblKq = new Label();                    // Nhãn kết quả
        this.add(lblKq);
        this.add(new Label());

        // Các nút nhấn
        btnCong = new
        Button("Cong");                        // Nút cộng
        btnCong.addActionListener(this
        );                                     // Bắt sự kiện
        this.add(btnCong);
        btnTru = new Button("Tru");            // Nút trừ
        btnTru.addActionListener(this);
        this.add(btnTru);
```

```

        btnNhan = new Button("Nhan"); // Nút nhân
        btnNhan.addActionListener(this);
        this.add(btnNhan);
        btnChia = new Button("Chia"); // Nút chia
        btnChia.addActionListener(this);
        this.add(btnChia);
        btnThoat = new Button("Thoat");// Nút thoát
        btnThoat.addActionListener(this);
        this.add(btnThoat);

        // Phương thức bắt sự kiện click vào nút đóng frame
        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }

    /* Phương thức xử lý sự kiện nút được nhấn */
    public void actionPerformed(ActionEvent ae){
        float x = Float.parseFloat(txt1.getText());
        float y = Float.parseFloat(txt2.getText());
        float kq = 0;
        if(ae.getSource() == btnCong) // Cộng hai số
            kq = x + y;
        if(ae.getSource() == btnTru) // Trừ hai số
            kq = x - y;
        if(ae.getSource() == btnNhan) // Nhân hai số
            kq = x*y;
        if(ae.getSource() == btnChia)&&(y != 0) // Chia
            hai số kq = x/y;
        if(ae.getSource() == btnThoat) // Thoát khỏi chương
            trình System.exit(0);

        // Thay đổi nội dung kết quả
        lblKq.setText("Ket qua la: " + String.valueOf(kq));
    }

```

```

public static void main(String[] args) {
    // Khai báo đối tượng demo
    EventDemo myFrame = new EventDemo();
    myFrame.setSize(300,150);           // Định kích cỡ frame
    myFrame.setVisible(true);           // Hiển thị frame
}
}

```

6.2 GIAO DIỆN VỚI CÁC ĐỐI TƯỢNG MULTIMEDIA

Nội dung phần này sẽ tập trung trình bày các đối tượng multimedia, bao gồm:

Ô đánh dấu (Checkbox) và Nút chọn (Radio button)

Lựa chọn (Choice)

Danh sách (List)

Trình đơn (Menu)

6.2.1 Ô đánh dấu và nút chọn

Checkbox và Radio button là các đối tượng dùng để đánh dấu, hoặc chọn thông tin. Sự khác biệt giữa chúng là checkbox cho phép chọn đồng thời nhiều ô cùng lúc, trong khi đó, trong mỗi nhóm radio button, chỉ cho phép chọn một thông tin.

Phương thức chung của hai lớp này:

setState(boolean)/getState(): Truy nhập đến trạng thái của nút.

Các phương thức khởi tạo Checkbox:

Checkbox(): Khởi tạo một ô đánh dấu rỗng.

Checkbox(String): Khởi tạo ô đánh dấu có nhãn xác định.

Checkbox(String, boolean): Khởi tạo ô đánh dấu có nhãn, có trạng thái xác định.

Các phương thức khởi tạo Radio button tương tự như Checkbox, ngoại trừ việc phải chỉ ra nhóm của các radio button:

Checkbox(String, boolean, CheckboxGroup);

Checkbox(String, CheckboxGroup, boolean);

Xử lý sự kiện thay đổi trạng thái nút chọn:

Kiểu sự kiện: ItemEvent

Cài đặt giao tiếp: ItemListener

Phương thức xử lý: itemStateChanged(ItemEvent)

Chương trình 6.8 minh họa việc dùng một nhóm radio button gồm ba nút, tương ứng với ba màu (RED, BLUE, GREEN). Khi click vào nút nào, thì màu nền sẽ đổi theo màu đó.

Chương trình 6.8

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;

public class RadioDemo extends Frame implements ItemListener{
    Checkbox cbxRed,
    cbxBlue, cbxGreen;

    public RadioDemo(){
        super("Radio demo!");
        //Chế độ hiển thị 3 dòng, 1 cột
        this.setLayout(new GridLayout(3,1));

        CheckboxGroup cbxg = new CheckboxGroup(); // Nhóm radio
        cbxRed = new Checkbox("Red", cbxg, true); // Nút red
        cbxRed.addItemListener(this); // Bắt sự kiện
        this.add(cbxRed);

        cbxBlue = new Checkbox("Blue", cbxg, false); // Nút blue
        cbxBlue.addItemListener(this); // Bắt sự kiện
        this.add(cbxBlue);

        cbxGreen = new Checkbox("Green", cbxg, false); // Nút green
        cbxGreen.addItemListener(this); // Bắt sự kiện
        this.add(cbxGreen);

        // Phương thức bắt sự kiện click vào nút đóng frame
        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }

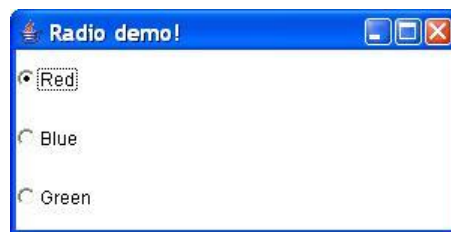
    /* Phương thức xử lý sự kiện thay đổi trạng thái nút */
    public void itemStateChanged(ItemEvent ie){
        if(ie.getStateChanged() == ItemEvent.SELECTED){
            String item = (String)ie.getItem();
            if(item.equals("Red")) // Đổi màu red
                this.setBackground(Color.red);
            if(item.equals("Blue")) // Đổi màu blue
                this.setBackground(Color.blue);
        }
    }
}
```

```

        this.setBackground(Color.blue);
        if(item.equals("Green"))                // Đổi màu green
            this.setBackground(Color.green);
        this.repaint();                        // Vẽ lại màu nền
    }
}

public static void main(String[] args) {
    // Khai báo đối tượng demo
    RadioDemo myFrame = new RadioDemo();
    myFrame.setSize(300,150);                // Định kích cỡ frame
    myFrame.setVisible(true);                // Hiển thị frame
}
}

```



Hình 6.7: Kết quả demo Radio Button

6.2.2 Lựa chọn

Choice là đối tượng menu xổ xuống, hiển thị một danh sách các item và cho phép người dùng chọn một trong số các item đó (tương tự đối tượng dropdown list của window). Các phương thức cơ bản của lớp Choice:

Choice(): Khởi tạo đối tượng choice.

addItem(String): Thêm một item vào danh sách lựa chọn.

remove(int): Xóa item ở vị trí thứ i trong danh sách (bắt đầu là vị trí 0).

removeAll(): Xóa toàn bộ item trong danh sách chọn.

select(int)/select(String): Chọn một item theo số thứ tự hoặc theo tên.

getSelectedIndex(): Trả về chỉ số của item được chọn.

getSelectedItem(): Trả về tên của item được chọn.

getItem(int): Trả về tên của item tương ứng với số thứ tự đưa vào.

Xử lý sự kiện thay đổi trạng thái nút chọn:

Kiểu sự kiện: ItemEvent

Cài đặt giao tiếp: ItemListener

Phương thức xử lý: itemStateChange(ItemEvent)

Chương trình 6.9 có chức năng tương tự như chương trình 6.8: Thay đổi màu nền theo màu được chọn. Nhưng trong chương trình này, ta dùng đối tượng choice, thay vì dùng radio button.

Chương trình 6.9

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
public class ChoiceDemo extends Frame implements ItemListener{ Choice myChoice;

    public ChoiceDemo(){
        super("Choice demo!");
        myChoice = new Choice();           // Khởi tạo
        myChoice.addItem("Red");           // Thêm item red
        myChoice.addItem("Blue");         // Thêm item blue
        myChoice.addItem("Green");        // Thêm item green
        myChoice.addItemListener(this); // Bắt sự kiện
        this.add(myChoice);                // Gắn vào frame
        // Phương thức bắt sự kiện click vào nút đóng frame
        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }

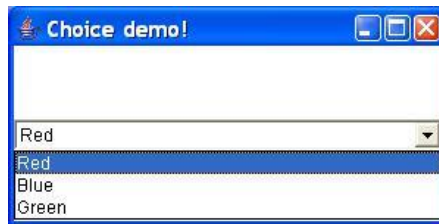
    /* Phương thức xử lý sự kiện thay đổi trạng thái item */ public void
    itemStateChange(ItemEvent ie){
        if(ie.getStateChanged() == ItemEvent.SELECTED){ String item =
            (String)ie.getItem();
            if(item.equals("Red")) // Đổi màu red this.setBackground(Color.red);
        }
    }
}
```

```

        if(item.equals("Blue"))                // Đổi màu blue
            this.setBackground(Color.blue);
        if(item.equals("Green"))                // Đổi màu green
            this.setBackground(Color.green);
        this.repaint();                        // Vẽ lại màu nền
    }
}

public static void main(String[] args) {
    // Khai báo đối tượng demo
    ChoiceDemo myFrame = new ChoiceDemo();
    myFrame.setSize(300,150);                // Định kích cỡ frame
    myFrame.setVisible(true);                // Hiển thị frame
}
}

```



Hình 6.8: Kết quả demo Choice Button

6.2.3 Danh sách

List là một danh sách hoạt động tương tự đối tượng choice. Tuy nhiên, list cho phép người dùng có thể chọn một hoặc nhiều item cùng một lúc. Các phương thức cơ bản của lớp List:

List(): Khởi tạo một danh sách rỗng, mỗi lần chỉ được chọn một item.

List(int): Tương tự, nhưng có qui định số dòng được nhìn thấy.

List(int, boolean): Khởi tạo một danh sách có số dòng được nhìn thấy xác định, chế độ cho phép chọn một hay nhiều item xác định bởi tham số thứ hai.

add(String): Thêm một item vào danh sách.

add(String, int): Chèn một item vào vị trí xác định trong danh sách. Nếu chỉ số chèn vượt ra khỏi phạm vi danh sách, item sẽ được thêm vào cuối.

replaceItem(String, int): Thay thế một item ở vị trí xác định (tham số thứ hai) trong danh sách bằng một item mới (tham số thứ nhất).

remove(int): Xoá item ở vị trí xác định trong danh sách.

removeAll(): Xoá toàn bộ item hiện có của danh sách.

getSelectedIndex(): Trả về index của item được chọn (danh sách đơn chọn).

getSelectedItem(): Trả về item được chọn (danh sách đơn chọn).

getSelectedIndexs(): Trả về chỉ số các item được chọn (danh sách đa chọn).

getSelectedItems(): Trả về các item được chọn (danh sách đa chọn).

Xử lý sự kiện khi thay đổi item được chọn:

Kiểu sự kiện: ItemEvent

Cài đặt giao tiếp: ItemListener

Phương thức xử lý: itemStateChange(ItemEvent);

Chương trình 6.10 minh hoạ việc sử dụng đối tượng list với khả năng đa chọn. Mỗi khi thay đổi item được chọn, một thông báo các màu được chọn sẽ hiện ra.

Chương trình 6.10

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
public class ListDemo extends Frame implements
    ItemListener{ List myList;
    Label lbl;

    public ListDemo(){
        super("List demo!");
        // Khởi tạo list đa chọn, chỉ nhìn được một dòng
        myList = new List(1, true);
        myList.setSize(300,150);
        // Thêm các item là các loại màu sắc
        myList.add("White");
        myList.add("Red");
        myList.add("Orange");
        myList.add("Green");
        myList.add("Yellow");
        myList.add("Blue");
```

```

myList.add("Black");
myList.addItemListener(this);    // Bắt sự kiện
this.setLayout(new FlowLayout());
this.add(myList);                // Gắn vào frame
lbl = new Label();               // Khởi tạo nhãn
this.add(lbl);                  // Gắn vào frame
// Phương thức bắt sự kiện click vào nút đóng frame
this.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
}
/* Phương thức xử lý sự kiện thay đổi trạng thái
item */ public void itemStateChanged(ItemEvent
ie){
    if((ie.getStateChange() == ItemEvent.SELECTED)||
        (ie.getStateChange() ==
            ItemEvent.DESELECTED)){
        String kq = "Cac mau duoc chon.";
        String[] items = myList.getSelectedItems();
        for(int i=0; i<items.length; i++)
            kq += items[i] + ", ";
        lbl.setText(kq);
    }
}

public static void main(String[] args) {
    // Khai báo đối tượng demo
    ListDemo myFrame = new
    ListDemo();
    myFrame.setSize(300,150);    Định kích cỡ frame
    myFrame.setVisible(true);    Hiển thị frame
}
}

```



Hình 6.9: Kết quả demo Listbox

6.2.4 Trình đơn

Trình đơn (menu) được dùng trên các thanh công cụ của các cửa sổ hoặc là popup menu xuất hiện khi ta click chuột phải vào một đối tượng. Java cung cấp một số lớp trình đơn:

MenuBar: Thanh trình đơn

Menu: Trình đơn đồ xuống

PopupMenu: Trình đơn xuất hiện khi click chuột phải.

MenuItem: Các mục chọn của trình đơn.

MenuBar

MenuBar là thanh công cụ dùng để chứa các trình đơn menu. Các phương thức cơ bản của lớp MenuBar:

MenuBar(): Khởi tạo một thanh công cụ cho trình đơn

add(Menu): Thêm một trình đơn mune lên thanh trình đơn menubar. Để đặt một menubar của một frame, ta gọi phương thức của frame:

<Đối tượng frame>.setMenuBar(<Đối tượng menubar>);

Menu và PopupMenu

Trình đơn menu là đối tượng sẽ xổ xuống khi click chuột lên đối tượng hiển thị của menu. Menu còn được gọi là menu con của một thanh trình đơn. Các phương thức cơ bản của lớp Menu:

Menu(String): Khởi tạo một menu, với tên xác định.

add(MenuItem): Thêm một item vào menu

add(Menu): Thêm một menu con vào menu đã có, dùng khi muốn tạo menu có nhiều mức.

addSeparator(): Thêm một đường phân vùng vào menu (để nhóm các item với nhau).

Xử lý sự kiện của lớp Menu:

Kiểu sự kiện: `ActionEvent`

Giao tiếp cài đặt: `ActionListener`

Phương thức xử lý: `actionPerformed(ActionEvent);`

MenuItem

MenuItem là đối tượng item trong các trình đơn menu. Mỗi item, khi được click vào sẽ có tác dụng như một nút lệnh. Các phương thức cơ bản của lớp MenuItem:

`MenuItem(String):` Khởi tạo một item.

`CheckboxMenuItem(String):` Khởi tạo một item có mục chọn như checkbox.

`getState():` Trả về trạng thái của item. Chỉ dùng cho item có mục chọn.

`enable():` Cho phép item hoạt động (là chế độ mặc định).

`disable():` Không cho phép item hoạt động (làm mờ item đi).

Xử lý sự kiện của lớp MenuItem:

Kiểu sự kiện: `ActionEvent`

Giao tiếp cài đặt: `ActionListener`

Phương thức xử lý: `actionPerformed(ActionEvent);`

Chương trình 6.11 minh họa việc sử dụng các loại menu:

Tạo một menubar của frame

Trên menubar, tạo một menu “File”. Khi click vào sẽ sổ xuống một menu với các item: New, Open, Save, Save As, Exit.

Khi bấm chuột phải vào frame, sẽ sổ ra một menu popup gồm các item: Cut, Copy, Paste.

Khi click chuột vào item nào trên các menu, một nhãn trong frame sẽ hiển thị tên của item vừa được chọn.

Chương trình kết thúc khi click vào item Exit.

Chương trình 6.11

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
public class MenuDemo extends Frame
    implements ActionListener, MouseListener{
    Menubar myBar;
    Menu myMenu;
    PopupMenu myPopup;
    Label lbl;
```

```

public MenuDemo(){
    super("Menu demo!");
    myBar = new Menubar();           // Thanh trình đơn
    this.setMenuBar(myBar);          // Thiết lập menubar của frame

    myMenu = new Menu("File");       // menu File
    myBar.add(myMenu);                // Gắn menu vào thanh trình đơn
    myMenu.addActionListener(this); // Bắt sự kiện
    myMenu.add(new MenuItem("New"));
    myMenu.add(new MenuItem("Open"));
    myMenu.addSeparator();           // Thêm dấu nhóm item
    myMenu.add(new MenuItem("Save"));
    myMenu.add(new MenuItem("Save As"));
    myMenu.addSeparator();
    myMenu.add(new MenuItem("Exit"));

    myPopup = new PopupMenu("Options");// Menu
    popup myPopup.addActionListener(this); // Bắt sự
    kiện myPopup(new MenuItem("Cut"));
    myPopup(new MenuItem("Copy"));
    myPopup(new MenuItem("Paste"));

    lbl = new Label();               // Khởi tạo nhãn
    this.add(lbl);                   // Gắn vào frame

    // Phương thức bắt sự kiện click vào nút đóng frame
    this.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
}

/* Phương thức xử lý sự kiện */
public void actionPerformed(ActionEvent ae){
    if(ae.getActionCommand().equals("Exit")){
        System.exit(0);
    }
}

```

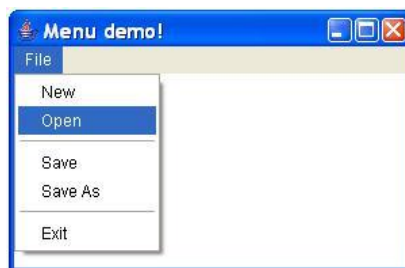
```

    }
    lbl.setText(ae.getActionCommand());
}

public void mouseEntered(MouseEvent me){ }    // Không xử lý
public void mouseExited(MouseEvent me){ }    // Không xử lý
public void mouseReleased(MouseEvent me){ }    // Không xử lý
public void mousePressed(MouseEvent me){ }    // Không xử lý
public void mouseClicked(MouseEvent me){
    myPopup.show(this, me.getX(), me.getY()); // Hiện menu popup
}

public static void main(String[] args) {
    // Khai báo đối tượng demo
    MenuDemo myFrame = new MenuDemo();
    myFrame.setSize(300,150);    // Định kích cỡ frame
    myFrame.setVisible(true);    // Hiển thị frame
}
}

```



Hình 6.10: Kết quả demo Menu

6.3 CÁC KỸ THUẬT TẠO TABLES

Nội dung phần này sẽ tập trung trình bày các kỹ thuật trình bày các đối tượng giao diện (component) trên frame theo các ý đồ thiết kế khác nhau bằng cách dùng bộ quản lý trình bày (Layout Manager). Bao gồm các kỹ thuật sau:

- Cách trình bày theo dòng (Flow layout)
- Cách trình bày theo mảng (Grid layout)
- Cách trình bày theo Border (Border layout)
- Cách trình bày theo GridBag (GridBag layout)
- Cách trình bày tự do (Null layout)

6.3.1 Trình bày Flow Layout

Cách trình bày Flow Layout sẽ xếp các đối tượng trên một hướng theo dòng. Nếu đối tượng mới thêm không đủ chỗ (chiều rộng) thì nó sẽ tự động thêm vào đầu dòng mới. Các phương thức:

FlowLayout(): Khởi tạo đối tượng trình bày.

FlowLayout(int): Khởi tạo đối tượng trình bày với cách căn lề xác định.

FlowLayout(int, int, int): Khởi tạo với ba tham số: Thứ nhất là cách căn lề, thứ hai là khoảng cách giữa hai dòng (chiều cao), thứ ba là khoảng cách giữa hai đối tượng (chiều ngang).

Tham số căn lề có thể nhận một trong ba giá trị:

FlowLayout.LEFT: Căn lề trái, là giá trị mặc định.

FlowLayout.CENTER: Căn lề giữa.

FlowLayout.RIGHT: Căn lề phải.

Chương trình 6.12 minh họa cách trình bày flow layout: Tạo ra một dãy 10 nút nhấn và gắn vào một frame theo kiểu flow layout.

Chương trình 6.12

```
package vidu.chuong6;
import java.awt.*;
public class FlowLayoutDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has somes buttons!");
        myFrame.setSize(300,150);           // Định kích cỡ frame
        myFrame.setLayout(new FlowLayout( ));// Thiết lập cách trình bày

        // Khai báo và khởi tạo button
        for(int i=0; i<10; i++)
            myFrame.add(new Button("Click"+i)); // Gắn vào frame
        myFrame.setVisible(true);           // Hiển thị frame
    }
}
```



Hình 6.11: Kết quả demo Flow layout

6.3.2 Trình bày Grid Layout

Cách trình bày Grid Layout sẽ sắp xếp các đối tượng theo dạng bảng, được xác định số hàng và số cột. Phương thức cơ bản:

`GridLayout(int, int)`: Khởi tạo một đối tượng trình bày. Hai tham số đầu vào lần lượt là số hàng và số cột của grid trình bày.

`GridLayout(int, int, int, int)`: Khởi tạo một đối tượng trình bày, hai tham số đầu xác định số hàng và số cột trình bày. Hai tham số sau xác định khoảng cách giữa các dòng và các cột của bảng.

Lưu ý:

Khi số lượng đối tượng được chèn nhiều hơn vào frame, ta muốn chương trình tự tính số hàng, hoặc tự tính số cột hiển thị, thì ta để tham số tương ứng là 0.

Ví dụ:

```
setLayout(new GridLayout(3,0));
```

sẽ cố định số hàng trình bày là 3, số cột là tùy thuộc vào số đối tượng trong frame.

```
setLayout(new GridLayout(0,2));
```

sẽ cố định số cột là 2, số dòng là mềm dẻo theo số các đối tượng trong frame.

Chương trình 6.13 minh họa cách trình bày grid layout: Tạo ra một dãy 10 nút nhấn và gắn vào một frame theo kiểu grid layout.

Chương trình 6.13

```
package vidu.chuong6;
import java.awt.*;
public class GridLayoutDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has some buttons!");
        myFrame.setSize(300,150);           // Định kích cỡ frame
```

```

myFrame.setLayout(new GridLayout(0,2));// Thiết lập cách trình bày

// Khai báo và khởi tạo button
for(int i=0; i<10; i++)
    myFrame.add(new Button("Click"+i));// Gắn vào frame
myFrame.setVisible(true);        // Hiển thị frame
}
}

```



Hình 6.12: Kết quả demo Grid layout

6.3.3 Trình bày Border Layout

Cách hiển thị Border Layout sẽ chia frame thành 5 vùng cố định và tự động kéo dãn các vùng sao cho chiếm hết bề mặt của frame:

West: Vùng phía tây, tức là phía lề bên trái.

East: Vùng phía đông, tức là phía lề bên phải.

North: Vùng phía bắc, tức là phía lề trên.

South: Vùng phía nam, tức là phía lề dưới.

Center: Vùng trung tâm, ở chính giữa frame.

Phương thức cơ bản của lớp BorderLayout:

BorderLayout(): Khởi tạo một đối tượng trình bày theo cách border.

Khi một frame được trình bày theo cách border, ta có thể dùng phương thức sau để gắn các đối tượng vào các vùng của frame:

<Đối tượng frame>.add(<Vùng border>, <Đối tượng component>);

Ví dụ:

```
myFrame.add("Center", new Button("Click"));
```

sẽ gắn vào vùng trung tâm của myFrame một nút nhấn có tên là "Click".

Lưu ý:

Cách trình bày border luôn chia frame thành 5 vùng xác định.

Nếu gắn nhiều đối tượng vào cùng một vùng, chỉ có đối tượng gắn sau là nhìn thấy được.

Nếu muốn trong một vùng chứa được nhiều đối tượng, ta có thể gắn vào mỗi vùng một Panel. Sau đó trong panel, ta chọn cách trình bày riêng cho panel và gắn các đối tượng vào panel.

Chương trình 6.14 minh hoạ cách trình bày border: Ta sẽ gắn vào năm vùng của frame năm nút nhấn khác nhau.

Chương trình 6.14

```
package vidu.chuong6;
import java.awt.*;
public class BorderLayoutDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has some buttons!");
        myFrame.setSize(300,150);           // Định kích cỡ frame
        myFrame.setLayout(new BorderLayout()); // Định cách trình bày

        // Khai báo và khởi tạo button
        myFrame.add("West", new Button("West")); // Gắn vào vùng west
        myFrame.add("East", new Button("East")); // Gắn vào vùng east
        myFrame.add("North", new Button("North")); // Gắn vào vùng north
        myFrame.add("South", new Button("South")); // Gắn vào vùng south
        // Gắn vào vùng center
        myFrame.add("Center", new Button("Center"));
        myFrame.setVisible(true);           // Hiển thị frame
    }
}
```



Hình 6.13: Kết quả demo Border layout

6.3.4 Trình bày GridBag Layout

Cách trình bày GridBag Layout cũng trình bày các đối tượng tương tự như Grid Layout: Các đối tượng sẽ được định vị theo vị trí các ô (cell) của một khung lưới (grid). Tuy

nhiên, GridBag cho phép ta định kích thước của đối tượng sẽ chiếm bao nhiêu ô và sẽ được đặt ở vị trí nào trong khung lưới. Các phương thức cơ bản:

GridBagLayout(): Khởi tạo một đối tượng trình bày theo cách gridbag.

setConstraints(Component, GridBagConstraints): Đặt vị trí và kích thước của đối tượng component theo các ràng buộc trong gridbagConstraints.

GridBagConstraints

Đây là lớp chứa các ràng buộc cho các đối tượng được trình bày theo cách GridBag. Các phương thức và thuộc tính cơ bản của lớp GridBagConstraints:

GridBagConstraints(): Khởi tạo một đối tượng ràng buộc của GridBag.

gridx/gridy: Vị trí của cell mà ta muốn đặt đối tượng vào (theo chiều X và chiều Y).

gridwidth/gridheight: Kích thước (vùng trình bày) của đối tượng (Theo chiều rộng và chiều cao).

fill: Xác định cách đặt đối tượng, theo 4 cách:

GridBagConstraints.NONE: Đối tượng không thay đổi kích thước theo các cell nó chiếm.

GridBagConstraints.VERTICAL: Đối tượng có chiều cao kín vùng nó chiếm

GridBagConstraints.HORIZONTAL: Đối tượng có chiều rộng kín vùng nó chiếm

GridBagConstraints.BOTH: Đối tượng có chiều cao và chiều rộng phủ kín vùng nó chiếm.

ipadx/ipady: Định đơn vị tăng giảm kích thước của đối tượng khi khung chứa bị thay đổi kích thước (theo chiều X và chiều Y).

insets: Xác định khoảng cách giữa các cell theo bốn hướng: Trên, dưới, trái, phải.

anchor: Xác định vị trí neo đối tượng khi kích thước khung chứa thay đổi. Bao gồm:
NORTH, NORTHEAST, NORTHWEST, EAST, SOUTH,
SOUTHEAST, SOUTHWEST.

weightx/weighty: Định khoảng cách lớn ra tương đối giữa các đối tượng với nhau.

Chương trình 6.15 minh họa cách trình bày gridbag: Tạo ra ba nút nhấn trong frame, mỗi nút có một số ràng buộc khác nhau về kích thước và vị trí.

Chương trình 6.15

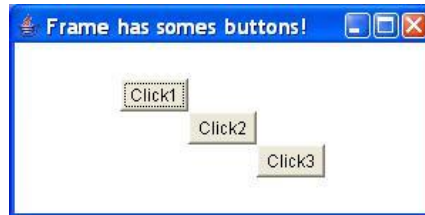
```
package vidu.chuong6;
import java.awt.*;
public class GridBagLayoutDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
        Frame myFrame = new Frame("Frame has somes buttons!");
        myFrame.setSize(300,150);           // Định kích cỡ frame
        GridBagLayout layout = new GridBagLayout();
        myFrame.setLayout(layout);          // Định cách trình bày
        // Khai báo đối tượng ràng buộc
        GridBagConstraints cts = new
        GridBagConstraints(); cts.fill =
        GridBagConstraints.BOTH;
        // Button1: vị trí (1,1), kích thước (1,1)
        Button btn1 = new Button("Click1");
        cts.gridx = 1;
        cts.gridy = 1;
        cts.gridheight = 1;
        cts.gridwidth = 1;
        layout.setConstraints(btn1, cts); // Định ràng buộc
        myFrame.add(btn1);                // Gắn vào frame
        // Button2: vị trí (2,2), kích thước
        // (1,1)
        Button btn2 = new
        Button("Click2");
        cts.gridx = 2;
        cts.gridy = 2;
        cts.gridheight = 1;
        cts.gridwidth = 1;
        layout.setConstraints(btn2, cts); // Định ràng buộc
        myFrame.add(btn2);                // Gắn vào frame
        Button btn3 = new Button("Click3");
        cts.gridx = 3;
        cts.gridy = 3;
```

```

cts.gridheight = 1;
cts.gridwidth = 1;
layout.setConstraints(btn3, cts); // Định ràng buộc
myFrame.add(btn3);                // Gắn vào frame

myFrame.setVisible(true);        // Hiển thị frame
}
}

```



Hình 6.14: Kết quả demo Gridbag layout

6.3.5 Trình bày Null Layout

Cách trình bày Null Layout sẽ trình bày các đối tượng không theo một quy tắc nào. Tất cả đều do người dùng tự định vị và thiết lập kích thước cho mỗi đối tượng.

Định vị đối tượng bằng phương thức setLocation():

```
<Đối tượng>.setLocation(Point);
```

Định kích thước đối tượng bằng phương thức setSize():

```
<Đối tượng>.setSize(int, int);
```

Ngoài ra, có thể vừa định vị, vừa định kích thước cho đối tượng thông qua phương thức:

```
<Đối tượng>.setBounds(int, int, int, int);
```

Trong đó, hai tham số đầu định vị đối tượng, hai tham số sau định kích thước đối tượng.

Chương trình 6.16 minh họa cách trình bày tự do Null layout: tạo ra hai nút nhấn và gắn vào frame theo hai cách khác nhau.

Chương trình 6.16

```

package vidu.chuong6;
import java.awt.*;
public class NullLayoutDemo{
    public static void main(String[] args) {
        // Khai báo và khởi tạo frame có tiêu đề
    }
}

```

```

Frame myFrame = new Frame("Frame has somes buttons!");
myFrame.setSize(300,150);           // Định kích cỡ frame
myFrame.setLayout(null);             // Định cách trình bày

// Button1: vị trí (10,30), kích thước (100,40)
Button btn1 = new Button("Click1");
btn1.setSize(100, 40);
btn1.setLocation(new Point(10, 30));
myFrame.add(btn1);                   // Gắn vào frame

// Button2: vị trí (70,120), kích thước (50,20)
Button btn2 = new Button("Click2");
btn2.setBounds(70, 120, 50, 20);
myFrame.add(btn2);                   // Gắn vào frame

myFrame.setVisible(true);            // Hiển thị frame
}
}

```



Hình 6.15: Kết quả demo Null layout

6.4 HTML & APPLET

Applet là một chương trình Java có thể chạy trong các trình duyệt web có hỗ trợ Java. Tất cả các applet đều là các lớp con của lớp Applet. Để tạo applet, ta cần import hai gói sau:

```

import java.applet.*;
import java.awt.*;

```

6.4.1 Cấu trúc của một Applet

Cấu trúc tổng quát của một applet như sau:

```

public class <Tên lớp applet> extends Applet{
    // Các thuộc
    tính public void
    init(){...}
    public void start(){...}
}

```



```

    public void stop(){...}
    public void destroy(){...}
    ... // Các phương thức khác
}

```

Các phương thức cơ bản của một applet:

init(): Khởi tạo các tham số, nếu có, của applet.

start(): Applet bắt đầu hoạt động.

stop(): Chấm dứt hoạt động của applet.

destroy(): Thực hiện các thao tác dọn dẹp trước khi thoát khỏi applet.

Lưu ý:

Không phải tất cả các applet đều phải cài đặt đầy đủ 4 phương thức cơ bản trên.

Applet còn có thể cài đặt một số phương thức tùy chọn (không bắt buộc) sau:

paint(Graphics): Phương thức vẽ các đối tượng giao diện bên trong applet. Các thao tác vẽ này được thực hiện bởi đối tượng đồ họa Graphics (là tham số đầu vào).

repaint(): Dùng để vẽ lại các đối tượng trong applet. Phương thức này sẽ tự động gọi phương thức update().

update(Graphics): Phương thức này được gọi sau khi thực hiện phương thức paint nhằm tăng hiệu quả vẽ. Phương này sẽ tự động gọi phương thức paint().

Chương trình 6.17 cài đặt một applet đơn giản, mỗi phương thức sẽ in ra thông báo rằng applet đang ở trong thời điểm tương ứng.

Chương trình 6.17

```

package vidu.chuong6;
import java.awt.*;
import java.applet.*;
public class SimpleApplet extends
Applet{
    private StringBuffer buffer;           // Chuỗi thông báo

    public void init(){                    // Khởi tạo
        buffer = new StringBuffer();
        addBuffer("initializing...");
    }

    public void start(){                    // Kích hoạt

```

```

        addBuffer("starting...");
    }
    public void stop(){                                // Dừng
        addBuffer("stopping...");
    }
    public void destroy(){                            // Thoát
        addBuffer("unloading...");
    }

    private void addBuffer(String
newBuffer){
        buffer.append(newBuffer);                    // Thêm thông báo
        repaint();
    }

    public void paint(Graphics g){
        g.drawString(buffer.toString(), 5,
        15);                                          // Hiện thông báo
    }
}

```

6.4.2 Sử dụng applet

Applet không thể chạy như một ứng dụng Java độc lập (nó không có hàm main), mà nó chỉ chạy được khi được nhúng trong một trang HTML (đuôi .htm, .html) và chạy bằng một trình duyệt web thông thường.

Các bước xây dựng và sử dụng một applet bao gồm:

Cài đặt chương trình có dạng một applet như mục 6.4.1

Biên dịch mã nguồn thành lớp .class

Nhúng mã .class của applet vào trang html.

Để nhúng một applet vào một trang html, ta dùng thẻ (tag) <Applet> như sau:

```

<APPLET CODE = "Tên_file_applet.class"
    WIDTH = "Chiều_rộng"
    HEIGHT = "Chiều_cao">
</APPLET>

```

Trong đó:

Tên applet là tên file mã nguồn đã biên dịch thành file chạy có đuôi .class của Java.

Chiều rộng và chiều cao là kích thước của vùng trên trang html mà applet sẽ được đặt vào.

Ví dụ, trong trang myHtml.htm có chứa nội dung như sau:

```
<HTML>
<HEAD>
  <TITLE> A simple applet </TITLE>
</HEAD>
<BODY>
  This is the output of applet:
  <APPLET CODE = "SimpleApplet.class" WIDTH=200
    HEIGHT=20> </APPLET>
</BODY>
</HTML>
```

sẽ nhúng applet đã được định nghĩa trong chương trình 6.17 vào một vùng có kích thước 200*20 trong trang myHtml. Bây giờ, ta có thể kiểm nghiệm chương trình applet của mình bằng cách mở trang myHtml trên các trình duyệt thông thường.

Chương trình 6.18 cài đặt một applet có chức năng tương tự như chương trình 6.7, thực hiện các thao tác tính toán cơ bản trên hai số. Ngoại trừ việc đây là một applet, nên có thể chạy trên một trang html.

Chương trình 6.18

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class AppletDemo extends Applet implements
  ActionListener{ Label lbl1, lbl2, lblKq;
  TextField txt1, txt2;
  Button btnCong, btnTru, btnNhan, btnChia, btnThoat;

  public void init(){
    this.setLayout(new GridLayout(6,2)); //Chế độ hiển thị 6 dòng, 2 cột
    lbl1 = new Label("So thu nhât:");// Nhãn số thứ nhất
    this.add(lbl1);
    txt1 = new TextField(); // Ô văn bản số thứ nhất
    this.add(txt1);
    lbl2 = new Label("So thu hai:");// Nhãn số thứ hai
    this.add(lbl2);
```

```

txt2 = new TextField();           // Ô văn bản số thứ hai
this.add(txt2);

lblKq = new Label();              // Nhãn kết quả
this.add(lblKq);
this.add(new Label());

// Các nút nhấn
btnCong = new Button("Cong"); // Nút cộng
btnCong.addActionListener(this); // Bắt sự kiện
this.add(btnCong);

btnTru = new Button("Tru");      // Nút trừ
btnTru.addActionListener(this);
this.add(btnTru);

btnNhan = new Button("Nhan"); // Nút nhân
btnNhan.addActionListener(this);
this.add(btnNhan);

btnChia = new Button("Chia"); // Nút chia
btnChia.addActionListener(this);
this.add(btnChia);

btnThoat = new Button("Thoat"); // Nút thoát
btnThoat.addActionListener(this);
this.add(btnThoat);
}

/* Phương thức xử lý sự kiện nút được nhấn */
public void actionPerformed(ActionEvent ae){
    float x = Float.parseFloat(txt1.getText());
    float y = Float.parseFloat(txt2.getText());
    float kq = 0;
    if(ae.getSource() == btnCong) // Cộng hai số
        kq = x + y;
    if(ae.getSource() == btnTru) // Trừ hai số
        kq = x - y;
    if(ae.getSource() == btnNhan) // Nhân hai số
        kq = x*y;
    if(ae.getSource() == btnChia)&&(y != 0) // Chia hai số kq = x/y;

```

```

        if(ae.getSource() == btnThoat)    // Thoát khỏi chương trình
            System.exit(0);
        // Thay đổi nội dung kết quả
        lblKq.setText("Ket qua la: " + String.valueOf(kq));
        repaint();                        // Vẽ lại các đối tượng
    }
}

```

Khi nhúng applet này vào một trang html bất kì, ta có thể kiểm tra thấy rằng nó có chức năng tương tự như ứng dụng 6.7.

Lưu ý, sự khác nhau giữa một application và một applet:

Application là một ứng dụng Java độc lập, nó có thể chạy độc lập trên máy ảo Java. Trong khi đó, applet chỉ chạy được khi nhúng trong một trang html, chạy nhờ vào các trình duyệt web có hỗ trợ Java.

Application chạy dựa vào hàm main(). Trong khi đó, applet không có hàm main().

Để hiển thị các thông báo, application dùng System.out.println(). Trong khi đó, applet dùng phương thức drawString() của lớp Graphics.

6.4.3 Truyền tham số cho Applet

Trong nhiều trường hợp, applet phải phụ thuộc vào các tham số ở bên ngoài truyền vào.

Khi đó ta có thể dùng thẻ PARAM của html để truyền tham số cho applet. Cú pháp:

```

<APPLET CODE = "Tên_file_applet.class"
    WIDTH = "Chiều_rộng"
    HEIGHT = "Chiều_cao">
    <PARAM NAME="Tên_biến" VALUE="Giá_trị">
    ... // Các tham số khác
</APPLET>

```

Khi đó, trong mã nguồn của applet, ta dùng phương thức getParameter() để đọc giá trị các tham số được truyền vào:

```
getParameter(Tên_biến);
```

Chương trình 6.19 minh họa việc truyền tham số cho một applet: Applet mô phỏng giao diện tìm kiếm: một nhãn hướng dẫn, một ô văn bản và một nút nhấn. Tùy vào ngôn ngữ mà nhãn và nút nhấn có giá trị text khác nhau. Biến ngôn ngữ là một tham số truyền từ trình duyệt vào. (Đây là mô phỏng giao diện, cơ chế tìm kiếm không hoạt động).

Chương trình 6.19

```

package vidu.chuong6;
import java.awt.*;
import java.applet.*;
public class ParamDemo extends Applet{
    Label lbl;
    TextField txt;
    Button btn;

    public void init(){
        this.setLayout(new GridLayout(2,2));    //Chế độ hiển thị 6 dòng, 2 cột
        String langue = getParameter("langue");// Loại ngôn ngữ
        if(langue.equals("vn")){                // Tiếng Việt
            lbl = new Label("Nhập từ khóa");    // Nhãn số thứ nhất
            btn = new Button("Tìm kiếm");    // Nút cộng
        }else if(langue.equals("fr")){          // Tiếng Pháp
            lbl = new Label("Tapez des mots");
            btn = new Button("Chercher");
        }else{                                  // Tiếng Anh, mặc định
            lbl = new Label("Enter keys");
            btn = new Button("Search");
        }
        txt = new TextField();
        this.add(lbl);
        this.add(txt);
        this.add(btn);
    }
}

```

Khi đó, applet phải được nhúng vào trang html với đoạn mã như sau:

```

<APPLET CODE = "ParamDemo.class" WIDTH = 200    HEIGHT = 20>
    <PARAM NAME="langue" VALUE="vn">
</APPLET>

```

Ta có thể thay thế value của param bằng các giá trị "vn", "fr" và "en" để thấy được các chế độ ngôn ngữ khác nhau được hiển thị trong applet.

6.5 GIỚI THIỆU VỀ SWING

Swing là thư viện lập trình mở rộng của Java. Nó mở rộng các đối tượng giao diện đồ họa cơ bản của Java. Swing còn được gọi là thư viện JFC (Java Foundation Class). Khi muốn sử dụng các đối tượng đồ họa của thư viện này, ta phải khai báo chỉ thị:

```
import javax.swing.*;
```

6.5.1 Mở rộng các đối tượng component

JFC mở rộng các đối tượng cơ bản của java thành các lớp tương ứng, ngoài trừ việc có thêm chữ “J” ở đầu mỗi tên lớp:

Button	□ JButton
Label	□ JLabel
TextField	□ JTextField
TextArea	□ JTextArea
Checkbox	□ JCheckbox
List	□ JList
Menu	□ JMenu

Các lớp mở rộng này có đầy đủ các phương thức của các đối tượng lớp cơ bản của thư viện java.awt. Ngoài ra, chúng được bổ sung một số phương thức tạo hiệu ứng giao diện.

Chương trình 6.20 minh họa việc sử dụng đối tượng JButton. Đối tượng JButton được mở rộng thêm một số tính năng sau:

JButton(String, Icon): Khởi tạo một nút nhấn với một tên nhãn và một ảnh nền. Ảnh nền có kiểu icon (tham số thứ hai).

setMnemonic(char): Định phím tắt cho nút lệnh. Khi người dùng nhấn “Ctrl+phím tắt” thì nút lệnh cũng thực thi tương tự như khi ta click chuột vào nút lệnh.

setBorder(new MatteBorder(int, int, int, int, Icon)): Thiết lập khung nền cho nút với các tham số: Khoảng cách từ chữ đến biên (độ rộng biên) theo các chiều trên dưới, trái phải, cuối cùng là ảnh nền cho nút.

setBorder(new LineBorder(int)): Thiết lập viền cho nút dạng hình chữ nhật, tham số xác định màu cho viền của nút. Ngoài ra, tham số của phương thức này còn có thể là các lớp SoftBevelBorder, EtchedBorder và TitleBorder.

setToolTipText(String): Thiết lập dòng tooltip cho đối tượng. Dòng này sẽ hiển ra khi ta di chuột lên đối tượng trên cửa sổ.

Chương trình 6.20

```
package vidu.chuong6;

import javax.swing.*;

public class JButtonDemo extends JFrame{
```

```

public static void main(String[] args) {
    // Khai báo và khởi tạo frame có tiêu đề
    JFrame myFrame = new JFrame("Frame has somes buttons!");
    myFrame.setSize(300,150);           // Định kích cỡ frame

    Giả sử ta có file ảnh myImage trong cùng thư
    mục Icon myIcon = new
    ImageIcon("myImage.gif");

    Button1: có nền là ảnh
    JButton btn1 = new JButton("Back Image", myIcon);
    // Gán tooltip cho nút
    btn1.setToolTipText("Button's background is an image");
    myFrame.getContentPane().add(btn1); // Gắn vào frame

    // Button2: có biên là ảnh
    JButton btn2 = new JButton("Border Image");
    // Gán tooltip cho nút
    btn1.setToolTipText("Button's border is an image");
    btn2.setBorder(new MatteBorder(10,10,10,10, myIcon));
    myFrame.getContentPane().add(btn2); // Gắn vào frame

    myFrame.setVisible(true);           // Hiển thị frame
}
}

```

Trong chương trình này, có dòng lệnh gắn các đối tượng vào frame bằng cách `getContentPane()`. Đây là phương thức mở rộng cho các đối tượng khung chứa container. Sự mở rộng này sẽ được trình bày chi tiết trong phần tiếp theo.

6.5.2 Mở rộng các đối tượng container

Tương tự như các đối tượng component, các đối tượng container cũng được mở rộng trong JFC thành các lớp có tên tương ứng và thêm kí tự "J" ở đầu:

```

Frame  □ JFrame
Panel  □□ JPanel
Dialog  □ JDialog

```

Chương trình 6.21 minh họa việc sử dụng các đối tượng mở rộng của khung chứa Frame thành JFrame. Khung chứa JFrame có nhiều tầng trình diễn khác nhau, các tầng là trong

suốt và chồng khít lên nhau, khiến cho ta vẫn có cảm giác các đối tượng được trình bày trên cùng một mặt phẳng như khung chứa Frame của thư viện chuẩn AWT.

Một số tầng hay sử dụng của lớp JFrame (theo thứ tự từ trong ra ngoài):

ContentPane: Là tầng thường dùng nhất, tầng này dùng để chứa các đối tượng component cơ bản như button, label, text, list...

MenuBarPane: Tầng dành để chứa các loại menu của frame như MenuBar, PopupMenu.

GlassPane: Tầng ngoài cùng, thường dùng để chứa các tooltip của các đối tượng trong tầng Content. Khi ta set tooltipText cho một đối tượng, tooltip đó sẽ tự động được add vào tầng Glass.

Để truy nhập vào một tầng bất kì, ta dùng phương thức có tên:

```
get + <Tên của tầng>();
```

Ví dụ:

```
JFrame myFrame = new JFrame("My JFrame");
myFrame.getContentPane().add("Center", new
JButton("Test"));
```

sẽ gắn một nút nhấn có nhãn Test vào tầng Content của khung chứa myFrame.

Chương trình 6.21 minh hoạ việc gắn các đối tượng vào các tầng khác nhau:

Gắn một nút nhấn vào tầng ContentPane.

Gắn một thanh MenuBar có chứa một menu File vào tầng MenuBarPane.

Chương trình 6.21

```
package vidu.chuong6;
import javax.swing.*;
import java.awt.event.*;

public class JFrameDemo extends JFrame implements
    ActionListener{ private JMenuBar myBar;
    private JMenu myMenu;

    public JFrameDemo(){
        super("JFrame demo");
        JButton btn = new JButton();

        Gắn nút nhấn vào tầng ContentPane
        this.getContentPane().add("Center",
            btn);
        myBar = new JMenuBar();
```

```

myMenu = new JMenu("File");
myMenu.add(new JMenuItem("Open"));
myMenu.add(new JMenuItem("New"));
myMenu.add(new JSeparator());
myMenu.add(new JMenuItem("Save"));
myMenu.add(new JMenuItem("Save As"));
myMenu.add(new JSeparator());
myMenu.add(new JMenuItem("Exit"));
myMenu.addActionListener(this);
myBar.add(myMenu);
// Gắn menubar vào tầng MenubarPane
this.getJMenuBar().add(myBar);

// Phương thức bắt sự kiện click vào nút đóng frame
this.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
}
/* Phương thức xử lý sự kiện */
public void actionPerformed(ActionEvent ae){
    if(ae.getActionCommand().equals("Exit")){
        System.exit(0);
    }
}

public static void main(String[] args) {
    // Khai báo và khởi tạo frame có tiêu đề
    JFrameDemo myFrame = new JFrameDemo();
    myFrame.setSize(300,150);        // Định kích cỡ frame
    myFrame.setVisible(true);        // Hiển thị frame
}
}

```

Lưu ý:

Vì các đối tượng mở rộng của thư viện JFC được bổ sung khá nhiều tính năng, đặc biệt là các tính năng đồ họa, do đó, các đối tượng này có nhược điểm là rất cồng kềnh. Vì lý do nặng tải, cho nên hiện nay, các đối tượng của thư viện JFC vẫn ít được phổ biến trong các ứng dụng applet.

6.6 CASE STUDY IV

Trong phần này, ta sẽ minh họa cách sử dụng các đối tượng đồ họa của thư viện chuẩn AWT để viết một chương trình mô phỏng một máy tính Calculator.

Tạo một frame làm khung chương trình, tiêu đề là “Java Calculator”

Phía trên là một Label (hoặc ô văn bản đều được, nhưng nếu dùng ô văn bản thì không cho edit) để hiển thị các số được nhập vào và kết quả tính toán.

Phía dưới là các nút nhấn tương ứng với các chữ số và phép toán. Nhưng để nhóm các nút nhấn cho đồng bộ và layout đẹp mắt, ta nhóm chúng vào một Panel.

Khi đó, frame sẽ chứa trực tiếp hai đối tượng: label và panel. Ta sử dụng layout kiểu null, và xác định vị trí chính xác cho label và panel.

Đối với Panel, ta cũng dùng GridLayout. Vì có 10 nút nhấn số và các nút nhấn toán tử: nút cộng, nút nhân, nút chia, nút trừ, nút căn bậc hai, nút phẩy thập phân, nút bằng, nút lũy thừa, nút nghịch đảo, nút reset. Nên sẽ tạo thành 4 dòng, 5 cột: mỗi dòng gồm có 3 nút số và hai nút chức năng:

Dòng 1: các nút 7, 8, 9, cộng, reset (C).

Dòng 2: các nút 4, 5, 6, trừ, lũy thừa.

Dòng 3: các nút 1, 2, 3, nhân, nghịch đảo.

Dòng 4: các nút 0, thập phân, nút bằng, nút chia, nút căn bậc hai.

Với các nút số và nút thập phân, khi click vào nút thì ký tự tương ứng được hiển thị lên trên label.

Với các nút chức năng, khi click vào thì thực hiện phép toán và hiển thị kết quả ra màn hình, nếu có.

Khi click vào nút bằng (kết quả) thì hiển thị kết quả trên label.

Chương trình 6.22 cài đặt chi tiết chương trình này.

Chương trình 6.22

```
package vidu.chuong6;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;
public class CalculatorDemo extends Frame implements ActionListener{
```

```

private boolean operatorState;           // Trạng thái của phép toán
private int operator;                   // Toán tử thực hiện
                                           // Số hạng
private float oldIterator;              trước
private Label lbl;
private Panel pnl;
private Button btn0, btn1, btn2, btn3, btn4, btn5, btn6,
    btn7, btn8, btn9, btnPoint, btnReset, btnAdd, btnSub,
    btnMul, btnDiv, btnPow, btnSqrt, btnRev, btnResult;

public CalculatorDemo(){
    super("Java Calculator");
    this.setSize(250, 250);
    this.setResizable(false);           // Không cho thay đổi size
    this.setLayout(null);               // Thiết lập layout

    lbl = new Label("0");                // Nhãn kết quả
    lbl.setAlignment(2);
    lbl.setSize(240, 30);
    lbl.setLocation(5, 25);
    this.add(lbl);

    Panel pnl = new Panel();             // Panel chứa các nút
    pnl.setSize(240, 190);
    pnl.setLocation(5, 55);
    pnl.setLayout(new GridLayout(4, 5)); // Thiết lập layout
    this.add(pnl);

    btn7 = new Button("7");              // Nút số 7
    btn7.addActionListener(this);        // Bắt sự kiện click chuột
    pnl.add(btn7);                       // Gắn vào panel

    btn8 = new Button("8");              // Nút số 8
    btn8.addActionListener(this);
    pnl.add(btn8);

    btn9 = new Button("9");              // Nút số 9
    btn9.addActionListener(this);

```

```

pnl.add(btn9);

btnAdd = new Button("+");           // Nút phép toán cộng
btnAdd.addActionListener(this);
pnl.add(btnAdd);

btnReset = new Button("C");         // Nút reset
btnReset.addActionListener(this);
pnl.add(btnReset);

btn4 = new Button("4");             // Nút số 4
btn4.addActionListener(this);
pnl.add(btn4);

btn5 = new Button("5");             // Nút số 5
btn5.addActionListener(this);
pnl.add(btn5);

btn6 = new Button("6");             // Nút số 6
btn6.addActionListener(this);
pnl.add(btn6);

btnSub = new Button("-");           // Nút phép toán trừ
btnSub.addActionListener(this);
pnl.add(btnSub);

btnPow = new Button("x^y");         // Nút phép toán lũy thừa
btnPow.addActionListener(this);
pnl.add(btnPow);

btn1 = new Button("1");             // Nút số 1
btn1.addActionListener(this);
pnl.add(btn1);

btn2 = new Button("2");             // Nút số 2
btn2.addActionListener(this);
pnl.add(btn2);

```

```

btn3 = new Button("3");           // Nút số 3
btn3.addActionListener(this);
pnl.add(btn3);

btnMul = new Button("*");         // Nút phép toán nhân
btnMul.addActionListener(this);
pnl.add(btnMul);

btnRev = new Button("1/x");
btnRev.addActionListener(this);   // Nút phép toán nghịch đảo
pnl.add(btnRev);

btn0 = new Button("0");
btn0.addActionListener(this);     // Nút số 0
pnl.add(btn0);

btnPoint = new Button(".");
btnPoint.addActionListener(this);
pnl.add(btnPoint);                // Nút dấu thập phân

btnResult = new Button("=");
btnResult.addActionListener(this);
pnl.add(btnResult);              // Nút kết quả

btnDiv = new Button("/");
btnDiv.addActionListener(this);
pnl.add(btnDiv);                 // Nút phép toán chia

btnSqrt = new Button("Sqrt");
btnSqrt.addActionListener(this);
pnl.add(btnSqrt);                // Nút phép toán căn bậc hai

operatorState = true;
operator = -1;
oldIterator = 0;
this.addWindowListener(new WindowAdapter(){

```

```

        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    });
}

public void actionPerformed(ActionEvent ae){
    float result;
    float newIterator = Float.parseFloat(lbl.getText());
    if(ae.getSource() == btnResult){
        switch(operator){
            case 0:
                result = oldIterator + newIterator;
                lbl.setText(String.valueOf(result));
                break;
            case 1:
                result = oldIterator - newIterator;
                lbl.setText(String.valueOf(result));
                break;
            case 2:
                result = oldIterator * newIterator;
                lbl.setText(String.valueOf(result));
                break;
            case 3:
                if(newIterator != 0){
                    result = oldIterator/newIterator;
                    lbl.setText(String.valueOf(result));
                }
                break;
            case 4:
                result = (float)Math.pow(oldIterator, newIterator);
                lbl.setText(String.valueOf(result));
                break;
        }
        operator = -1;
        operatorState = true;
    }
}

```

```

        return;
    }
    if(ae.getSource() == btnRev){
        newIterator = Float.parseFloat(lbl.getText());
        if(newIterator != 0){
            result = (float)1/newIterator;
            lbl.setText(String.valueOf(result));
        }
        operator = -1;
        operatorState = true;
        return;
    }
    if(ae.getSource() == btnSqrt){
        newIterator = Float.parseFloat(lbl.getText());
        if(newIterator >= 0){
            result = (float)Math.sqrt(newIterator);
            lbl.setText(String.valueOf(result));
        }
        operator = -1;
        operatorState = true;
        return;
    }
    if(ae.getSource() == btnPoint){
        lbl.setText(lbl.getText() + "." );
        return;
    }
    if(ae.getSource() == btnAdd){
        operator = 0;
        operatorState = true;
        oldIterator = Float.parseFloat(lbl.getText());
        return;
    }
    if(ae.getSource() == btnSub){
        operator = 1;
        operatorState = true;
        oldIterator = Float.parseFloat(lbl.getText());
        return;
    }

```



```

    }
    if(ae.getSource() == btnMul){
        operator = 2;
        operatorState = true;
        oldIterator = Float.parseFloat(lbl.getText());
        return;
    }
    if(ae.getSource() == btnDiv){
        operator = 3;
        operatorState = true;
        oldIterator = Float.parseFloat(lbl.getText());
        return;
    }
    if(ae.getSource() == btnPow){
        operator = 4;
        operatorState = true;
        oldIterator = Float.parseFloat(lbl.getText());
        return;
    }
    if(ae.getSource() == btnReset){
        operator = -1;
        operatorState = true;
        oldIterator = 0;
        lbl.setText("0");
        return;
    }
    // Trường hợp click vào nút số
    if(operatorState){ // Bắt đầu số mới
        lbl.setText(ae.getActionCommand());
        operatorState = false;
    }else // Gõ tiếp số cũ
        lbl.setText(lbl.getText() + ae.getActionCommand());
}

public static void main(String[] args) {
    // Khai báo và khởi tạo frame
    CalculatorDemo myFrame = new CalculatorDemo();

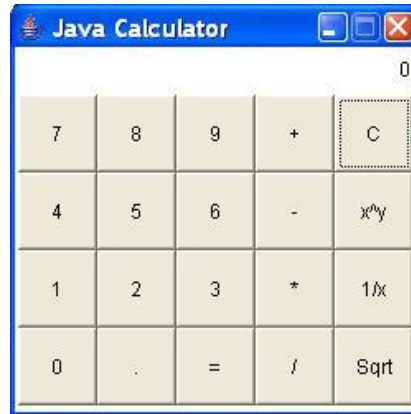
```

```

        myFrame.setVisible(true);           // Hiển thị frame
    }
}

```

Chương trình 6.22 sẽ cho kết quả như hình 6.16:



Hình 6.16: Kết quả demo case study

TỔNG KẾT CHƯƠNG 6

Nội dung chương 6 đã trình bày phương pháp lập trình giao diện với các đối tượng trong thư viện chuẩn AWT của Java:

Các đối tượng có chức năng làm vật chứa cho các đối tượng giao diện: Frame, Panel, Dialog.

Các đối tượng là thành phần giao diện: Label, Button, TextField, TextArea, Checkbox, List, Menu.

Phương pháp nắm bắt và xử lý các sự kiện đặc thù của từng đối tượng giao diện.

Các phương pháp trình bày trên các vật chứa: FlowLayout, GridLayout, BorderLayout, GridBagLayout, NullLayout.

Chương 6 cũng giới thiệu một phương pháp lập trình giao diện Java được nhúng trong các trang web, đó là lập trình applet:

Cách tạo ra một applet với các phương thức cơ bản.

Cách nhúng một applet vào một trang web.

Cách kiểm nghiệm một applet sau khi nhúng vào trang web bằng các trình duyệt.

Ngoài ra, chương này cũng giới thiệu cách sử dụng thư viện các đối tượng đồ họa mở rộng JFC của Java. Các đối tượng của thư viện JFC có chức năng hoàn toàn tương tự các đối tượng tương ứng trong thư viện chuẩn AWT. Ngoài ra, chúng còn được bổ sung thêm một số khả năng đồ họa cao cấp.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 6

Viết chương trình thay đổi màu nền của frame theo lựa chọn của người dùng:

Tạo ra các nút nhấn có tên theo các màu: Blue, Cyan, Gray, Green, Magenta, Orange, Pink, Red, White, Yellow.

Khi click chuột vào nút nào, màu nền của frame sẽ đổi theo màu đó.

Viết chương trình thay đổi màu nền trong bài 1 bằng ô văn bản. Tạo ra một ô văn bản duy nhất, khi người dùng gõ vào một trong số các màu trong bài 1 và gõ enter, màu nền của frame sẽ đổi theo màu đó. Nếu người dùng gõ sai màu, không làm gì cả.

Viết chương trình thay đổi màu nền trong bài 1 bằng nút chọn radio. Tạo một nhóm các nút radio tương ứng với các loại màu. Khi màu nào được chọn, màu nền của frame sẽ thay đổi theo màu đó.

Viết chương trình thay đổi màu nền trong bài 1 bằng danh sách chọn list. Tạo một List có các item tương ứng với các loại màu. Khi màu nào được chọn, màu nền của frame sẽ thay đổi theo màu đó.

Viết chương trình thay đổi màu nền trong bài 1 bằng menu. Tạo một menubar, trên đó có gắn một menu tên là color, khi click chuột vào menu color, sẽ xổ xuống các màu tương ứng trong bài 1. Khi màu nào được chọn, màu nền của frame sẽ thay đổi theo màu đó.

Viết chương trình thay đổi màu nền trong bài 1 bằng menu popup. Tạo một menu popup trong frame, khi click chuột phải lên frame, sẽ hiện ra menu gồm các màu tương ứng trong bài 1. Khi màu nào được chọn, màu nền của frame sẽ thay đổi theo màu đó.

Viết lại các chương trình trong các bài tập 1 đến 6 dưới dạng applet và nhúng chúng vào trang myHtml.html để chạy. (Trang này phải cùng thư mục với các lớp vừa cài đặt và biên dịch).

CHƯƠNG 7

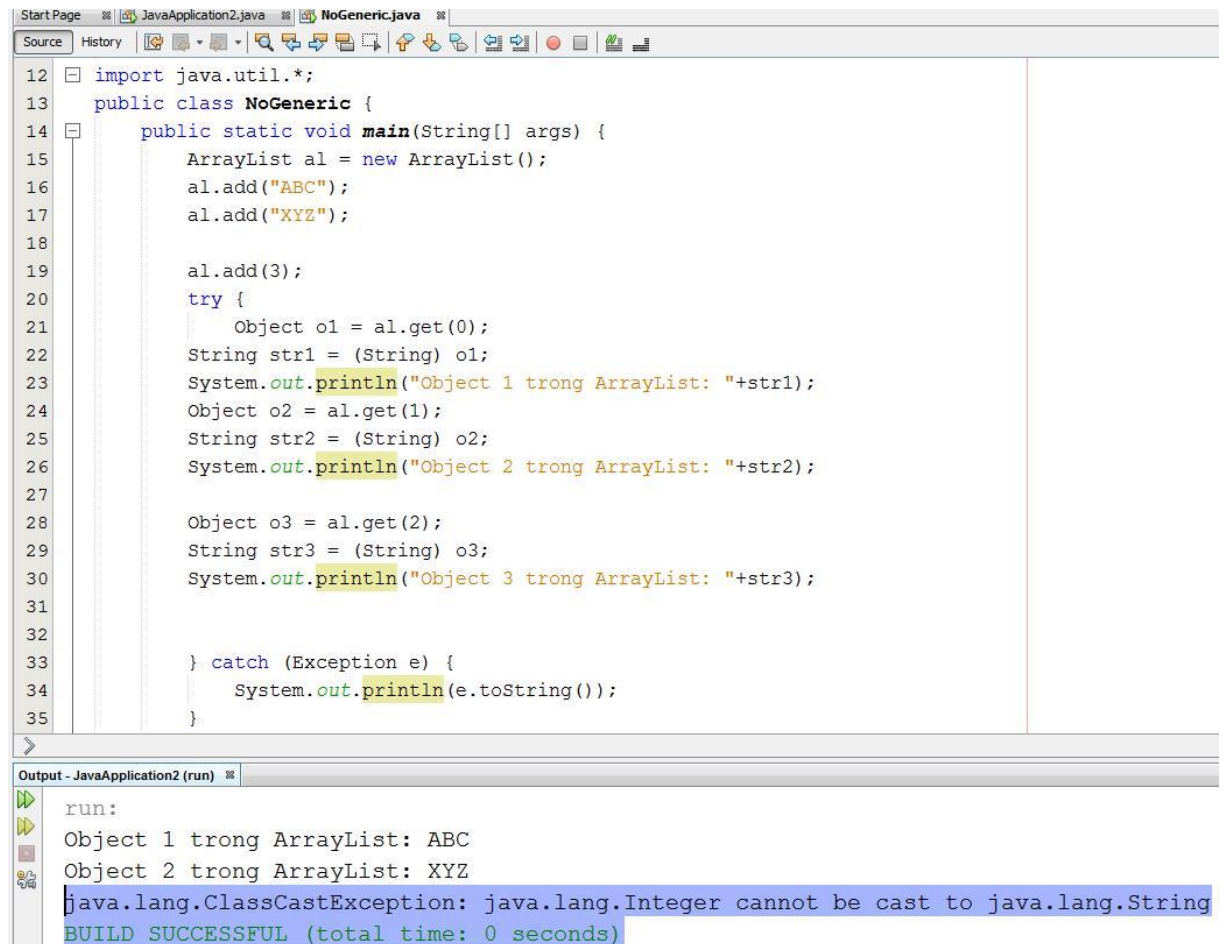
LẬP TRÌNH GENERIC

Chương này lập trình khái quát (Generic Programming). Định nghĩa, cách thức sử dụng và các trường hợp ứng dụng cụ thể.

7.1 Giới thiệu về Generic:

7.1.1 Đặt vấn đề

Tạo một ArrayList **al** với mục đích chỉ dùng để chứa phần tử có kiểu String , tuy nhiên ta lại cố gắng thêm vào danh sách al phần tử không phải kiểu String , như vậy khi ta ép kiểu về String để in phần tử đó ra thì sẽ có ngoại lệ vì chắc chắn số nguyên không thể ép kiểu về String (xem ví dụ)



```
12 import java.util.*;
13 public class NoGeneric {
14     public static void main(String[] args) {
15         ArrayList al = new ArrayList();
16         al.add("ABC");
17         al.add("XYZ");
18
19         al.add(3);
20         try {
21             Object o1 = al.get(0);
22             String str1 = (String) o1;
23             System.out.println("Object 1 trong ArrayList: "+str1);
24             Object o2 = al.get(1);
25             String str2 = (String) o2;
26             System.out.println("Object 2 trong ArrayList: "+str2);
27
28             Object o3 = al.get(2);
29             String str3 = (String) o3;
30             System.out.println("Object 3 trong ArrayList: "+str3);
31
32         } catch (Exception e) {
33             System.out.println(e.toString());
34         }
35     }
36 }
```

Output - JavaApplication2 (run)

```
run:
Object 1 trong ArrayList: ABC
Object 2 trong ArrayList: XYZ
java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String
BUILD SUCCESSFUL (total time: 0 seconds)
```

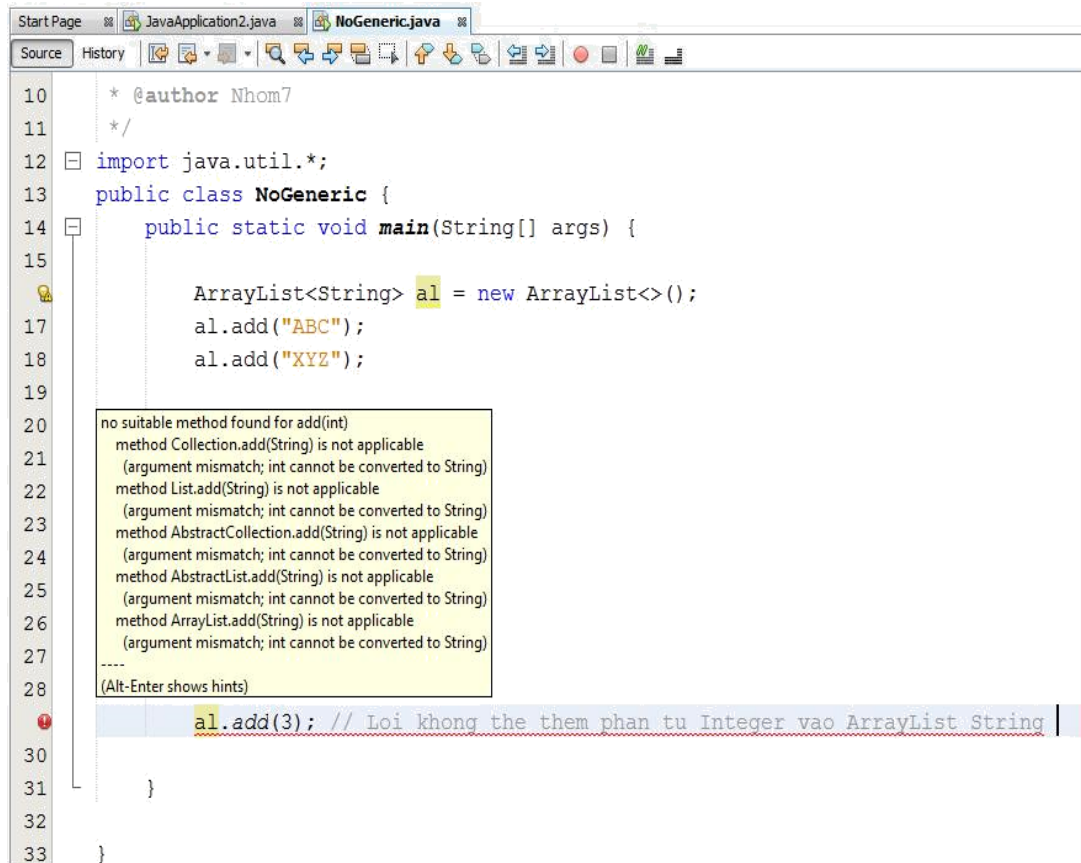
Hình 7.1: Ví dụ ép kiểu sai

7.1.2. Khái niệm Generic

Java 5 đưa vào khái niệm **Generics**.

Thuật ngữ “Generics” nghĩa là tham số hóa kiểu dữ liệu. Tham số hóa kiểu dữ liệu rất quan trọng vì nó cho phép chúng ta tạo ra và sử dụng một class, interface, method với nhiều kiểu dữ liệu khác nhau.

Với sự trợ giúp của **Generics**, ta có thể tạo ra một đối tượng **ArrayList** chỉ cho phép chứa các phần tử có kiểu String, và không cho phép chứa các phần tử có kiểu khác. Một class, interface hay một method mà thực hiện trên một kiểu tham số xác định thì gọi là Generic.



Hình 7.2: Sử dụng Generic

Khi ta tạo đối tượng `ArrayList<String>`, nó chỉ cho phép chứa các phần tử dạng `String`, trình biên dịch Java không cho phép chứa phần tử kiểu khác, giúp chúng ta có thể biết được lỗi đó là lỗi cú pháp, ta sẽ dễ dàng sửa chữa hơn.

Hơn nữa ở ví dụ trên, ta phải viết rõ ràng việc ép kiểu để chuyển các phần tử từ `Object` về kiểu thực của chúng, điều trên thực sự là hơi bất tiện và không cần thiết với một ngôn ngữ.

7.2 Các phương thức Generic trong Java:

Phương thức Generic trong Java: là cách thức tạo ra một phương thức mà có thể được gọi với nhiều kiểu dữ liệu khác nhau. Dựa vào kiểu dữ liệu được truyền vào, trình biên dịch sẽ xử lý mỗi lời gọi phương thức sao cho phù hợp.

Ký tự	Ý nghĩa
E	Element – phần tử
K	Key – khóa
V	Value – giá trị

T	Type – kiểu dữ liệu
N	Number – số

Bảng 7.1: Các ký hiệu phổ biến trong Generics

Ta có thể viết một khai báo phương thức generic đơn mà có thể được gọi với các tham số của các kiểu khác nhau. Dựa trên các kiểu tham số được truyền tới phương thức generic này, bộ biên dịch xử lý mỗi lần gọi phương thức một cách thích hợp. Dưới đây là các qui tắc để định nghĩa các phương thức Generic:

Tất cả khai báo phương thức generic có một khu vực tham số kiểu được giới hạn bởi các dấu ngoặc nhọn (< và >) mà đứng trước kiểu trả về của phương thức (< E > trong ví dụ sau đây).

Mỗi khu vực tham số kiểu chứa một hoặc nhiều tham số kiểu phân biệt nhau bởi dấu phẩy. Một tham số kiểu, cũng được biết như là biến kiểu, là một định danh mà xác định một tên kiểu generic.

Các tham số kiểu có thể được sử dụng để khai báo kiểu trả về và hoạt động như là nơi giữ (placeholder) cho các kiểu của các tham số được truyền tới phương thức generic, mà được biết như là các tham số kiểu thực sự.

Phần thân phương thức generic được khai báo giống như bất kỳ phương thức nào khác. Ghi chú rằng các tham số kiểu chỉ có thể biểu diễn các kiểu tham chiếu, không phải là các kiểu gốc (như int, double, và char).

Ví dụ sử dụng phương thức Generic đơn để in ra các phần tử kiểu khác nhau (ArrayList<Integer> và ArrayList<String>)

```

12 import java.util.*;
13 public class VD1 {
14     public static void main(String[] args) {
15         ArrayList<Integer> in = new ArrayList();
16         ArrayList<String> al = new ArrayList();
17         in.add(1);
18         in.add(2);
19         al.add("ABC");
20         al.add("XYZ");
21         System.out.println("In ra cac phan tu ");
22         System.out.println("-----Integer-----");
23         xuat(in);
24         System.out.println("-----String-----");
25         xuat(al);
26     }
27     public static <E> void xuat(ArrayList<E> input) {
28         //System.out.print("\n");
29         for(E x :input) {
30             System.out.print(x + " ");
31         }
32         System.out.println("");
33     }
}

```

Output - JavaBasic (run) x

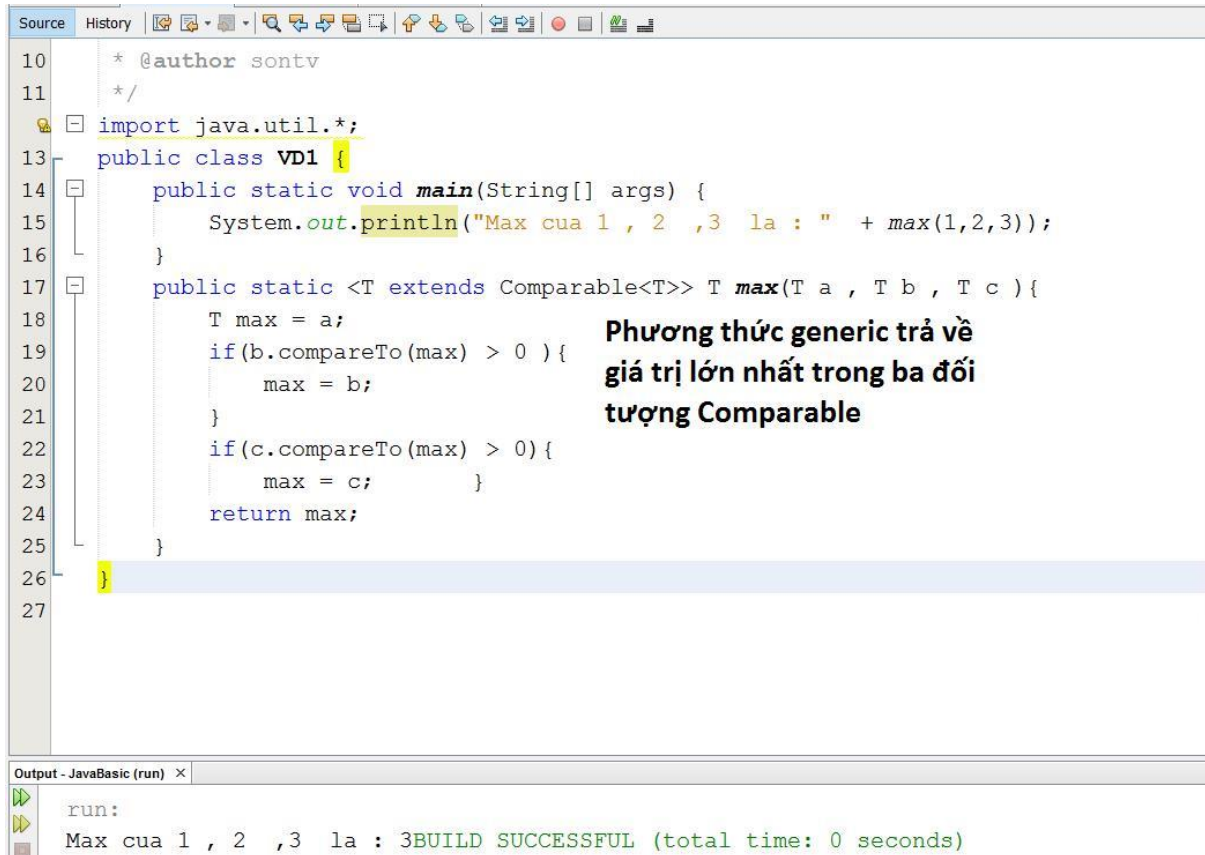
```

run:
In ra cac phan tu
-----Integer-----
1 2
-----String-----
ABC XYZ
BUILD SUCCESSFUL (total time: 0 seconds)

```

Hình 7.3: Phương thức Generic đơn

Ví dụ về phương thức generic trả về giá trị lớn nhất trong ba đối tượng



```
10  * @author sontv
11  */
12  import java.util.*;
13  public class VD1 {
14      public static void main(String[] args) {
15          System.out.println("Max của 1 , 2 ,3 la : " + max(1,2,3));
16      }
17      public static <T extends Comparable<T>> T max(T a , T b , T c ){
18          T max = a;
19          if(b.compareTo(max) > 0 ){
20              max = b;
21          }
22          if(c.compareTo(max) > 0 ){
23              max = c;
24          }
25          return max;
26      }
27  }
```

Phương thức generic trả về giá trị lớn nhất trong ba đối tượng Comparable

Output - JavaBasic (run) x

run:
Max của 1 , 2 ,3 la : 3BUILD SUCCESSFUL (total time: 0 seconds)

Hình 7.4: Phương thức Generic có extends

7.3 Các lớp Generic Classes

Các phần trước đã nói đến kiểu `ArrayList<T>`. Đây là Collection có thể là tập hợp của đối tượng bất kỳ ví dụ:

`ArrayList<String>` hoặc `ArrayList<Student>` hoặc `ArrayList<Object>` ...

Như các bạn đã thấy cách tạo ra lớp Generic Class `ArrayList<T>` có thể sử dụng cho các đối tượng khác nhau và ở đây mình sẽ nói cho các bạn cách tạo ra và sử dụng nó giống như trên.

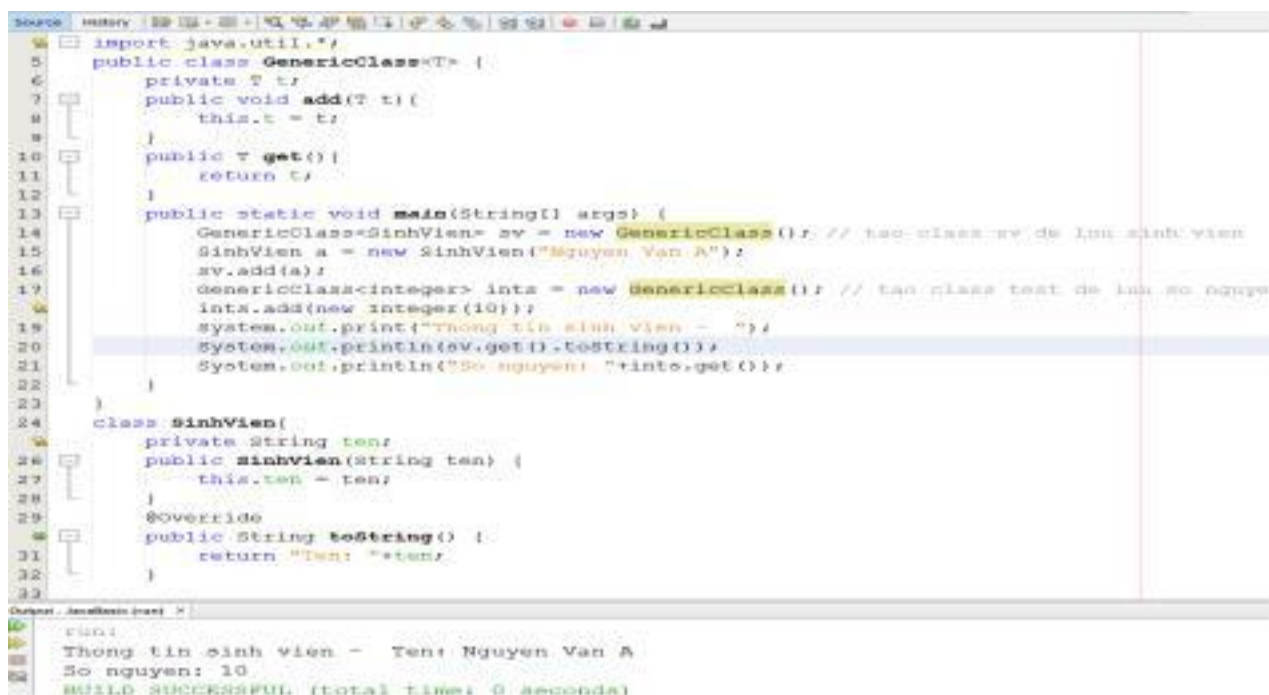
Lớp Generic là một cơ chế để chỉ rõ mối quan hệ giữa Lớp và kiểu dữ liệu liên quan đến nó (type parameter).

“Các Tham số kiểu” sẽ được xác định tại thời điểm đối tượng của Lớp được tạo

Quy ước về tên của Tham số kiểu cũng giống như quy ước dùng trong Phương thức Generic ở phần trên

Lớp Generic trông giống như một khai báo lớp không phải Generic, ngoại trừ là tên lớp được theo sau bởi một khu vực tham số kiểu.

Như với phương thức Generic, khu vực tham số kiểu của một lớp Generic có thể có một hoặc nhiều tham số kiểu phân biệt nhau bởi dấu phẩy. Những lớp này còn được biết như là các lớp tham số hóa hoặc các kiểu tham số hóa bởi vì chúng chấp nhận một hoặc nhiều tham số. Ví dụ về sử dụng class với Generic trong hình 7.5.



```
1 import java.util.*;
2
3 public class GenericClass<T> {
4     private T t;
5     public void add(T t) {
6         this.t = t;
7     }
8     public T get() {
9         return t;
10    }
11
12    public static void main(String[] args) {
13        GenericClass<SinhVien> sv = new GenericClass(); // tạo class sv để lưu sinh viên
14        SinhVien a = new SinhVien("Nguyễn Văn A");
15        sv.add(a);
16        GenericClass<Integer> ints = new GenericClass(); // tạo class test để lưu số nguyên
17        ints.add(new Integer(10));
18        System.out.println("Thông tin sinh viên - ");
19        System.out.println(sv.get().toString());
20        System.out.println("Số nguyên: "+ints.get());
21    }
22 }
23
24 class SinhVien {
25     private String ten;
26     public SinhVien(String ten) {
27         this.ten = ten;
28     }
29     @Override
30     public String toString() {
31         return "Ten: "+ten;
32     }
33 }
```

Output: JavaSE10 (run) 7

```
run:
Thông tin sinh viên - Ten: Nguyễn Văn A
Số nguyên: 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Hình 7.5: Ví dụ khai báo lớp Generics

Khi ta tạo ra class tương tự như trên nhưng kiểu dữ liệu sẽ để dạng động (nghĩa là nó có thể chứa bất cứ thứ gì). Cách sử dụng sẽ đa năng hơn, ta có thể tạo ra Queue chứa String hoặc Int hoặc các đối tượng khác nhau.

Xem ví dụ ta thấy trong định nghĩa class GenericClass đều làm việc kiểu dữ liệu trừu tượng T , và có thể thay thế bất kỳ kiểu dữ liệu nào. Để làm được điều này trong quá trình định nghĩa ta phải đảm bảo các thuộc tính, phương thức hoạt động đúng trên nhiều kiểu dữ liệu khác nhau, giảm thiểu trường hợp phát sinh lỗi...

Ví dụ sử dụng lớp với Generics để sắp xếp bất kỳ ArrayList với kiểu dữ liệu nào:

```

15 public class VD1 {
16     public static void main(String[] args) {
17         Scanner in1 = new Scanner(System.in);
18         ArrayList<Integer> in = new ArrayList<>();
19         System.out.println("Moi nhap so phan tu :");
20         int n = in1.nextInt();
21         for(int i = 0 ; i < n ; i++){
22             int x = in1.nextInt();
23             in.add(x);
24         }
25         System.out.println("Mang vua nhap:");
26         for(int i = 0 ; i < in.size() ; i++){
27             System.out.print(in.get(i) + " ");
28         }
29         System.out.println();
30         sort(in);
31         System.out.println("Mang sap xep su dung lop generics ");
32         for(int i = 0 ; i < in.size() ; i++){
33             System.out.print(in.get(i) + " ");
34         }
35     }
36     public static <T extends Comparable<T>> void sort(ArrayList<T> a){
37         for(int i = 0 ; i < a.size() -1 ; i++){
38             for(int j = i + 1 ; j < a.size() ; j++){
39                 if(a.get(i).compareTo(a.get(j)) > 0) {
40                     T tmp = a.get(i);
41                     a.set(i, a.get(j));
42                     a.set(j, tmp);
43                 }
44             }
45         }
46     }
47 }

```

Output - JavaBasic (run) ×

```

Mang vua nhap:
2 4 1 6 8 2
Mang sap xep su dung lop generics
1 2 2 4 6 8 BUILD SUCCESSFUL (total time: 5 seconds)

```

Hình 7.6: Sử dụng lớp Generic

Ví dụ sử dụng nhiều loại class được ghép cặp với nhau

```

46
47
48
49 class SinhVien{
50     private String ten;
51     public SinhVien(String ten) {
52         this.ten = ten;
53     }
54     public String toString(){
55         return ten;
56     }
57 }
58 class MonHoc{
59     private String tenMH;
60     public MonHoc(String tenMH) {
61         this.tenMH = tenMH;
62     }
63     @Override
64     public String toString() {
65         return tenMH;
66     }
67 }

```

```

12 import java.util.ArrayList;
13 public class Pair <T,S>{
14     private T firt;
15     private S second;
16     public Pair(T firt, S second){
17         this.firt = firt;
18         this.second = second;
19     }
20     public T getFirt(){
21         return firt;
22     }
23     public S getSecond(){
24         return second;
25     }
26     public void display() {
27         System.out.println(firt + "\t" + second);
28     }
29     public String toString(){
30         return (firt + "\t" + second);
31     }
32     public static void main(String[] args) {
33         ArrayList<SinhVien> sv = new ArrayList<>();
34         ArrayList<MonHoc> mh = new ArrayList<>();
35         SinhVien a = new SinhVien("Nguyen van A");
36         sv.add(a);
37         MonHoc b = new MonHoc("JAVA");
38         mh.add(b);
39         ArrayList<Pair<SinhVien,MonHoc>> kq = new ArrayList<>();
40         Pair<SinhVien,MonHoc> k = new Pair(sv.get(0) , mh.get(0));
41         kq.add(k);
42         System.out.println("Sau khi ghép cặp là:");
43         System.out.print("TEN\t\tMONHOC");
44         System.out.println();
45         System.out.println(kq.get(0));
46

```

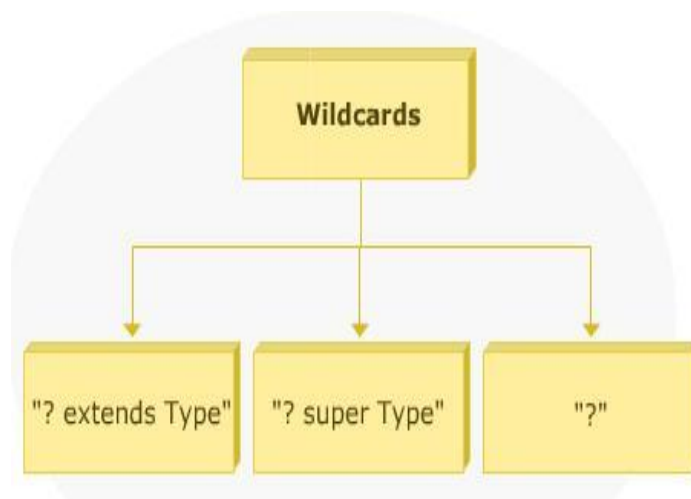
```

Output - JavaBasic (run) x
run:
Sau khi ghép cặp là:
TEN          MONHOC
Nguyen van A  JAVA
BUILD SUCCESSFUL (total time: 0 second

```

Hình 7.7: Sử dụng các lớp Generic để ghép cặp

7.4 Các kí tự đại diện Generics



Hình 7.8: Các dạng ký tự đại diện

“ ? ” : Đại diện cho một kiểu chưa xác định.

```

public class VD0{
    public static void main(String[] args) {
        private boolean check1(RestricExample<T> e) {
            if(number.doubleValue() == e.number.doubleValue()) {
                return true;
            }
            return false;
        }

        private boolean check2(RestricExample<?> e) {
            if(number.doubleValue() == e.number.doubleValue()) {
                return true;
            }
            return false;
        }
    }
}

```

Hình 7.9: Sử dụng dấu ?

Phương thức check1 chỉ chấp nhận tham số e có kiểu dữ liệu giống với biến number. Phương thức check2 chấp nhận tham số e có kiểu dữ liệu khác với biến number. * “? Extends Type” : chấp nhận bất kỳ đối tượng nào miễn là đối tượng này kế thừa từ type hoặc đối tượng của type.

<? super type> chấp nhận bất kỳ đối tượng nào miễn là đối tượng này là cha của type hoặc đối tượng của type.

```

13 public class VD4 {
14     public static void main(String[] args) {
15         Stack<String> st1 = new Stack<>();
16         Stack<Object> st2 = new Stack<>();
17         st2.push("ABC");
18         st2.push(2);
19         st1.push("CBA");
20         add(st1,st2);
21         System.out.println(st2);
22     }
23     public static <T> void add(Stack<T> st1,Stack<? super T> st2){
24         while(!st1.isEmpty()){
25             st2.push(st1.pop());
26         }
27     }
28 }
29
Output - JavaBasic (run) x
run:
[ABC, 2, CBA]
BUILD SUCCESSFUL (total time: 0 seconds)

```

Hình 7.9: Sử dụng ? super

7.5 Ưu và một số lưu ý khi sử dụng Generics.

7.5.1 Ưu điểm

Kiểm tra dữ liệu chặt chẽ ở Compile – time mà không phải là Runtime – error. Nên chúng ta dễ dàng kiểm soát lỗi hơn.

Hạn chế việc ép kiểu thủ công mà không an toàn.

Giúp chúng ta viết được các thuật toán được sử dụng nhiều , dễ dàng thay đổi , an toàn dữ liệu và dễ đọc hơn , là mục tiêu của ai lập trình OPP đều muốn hướng tới.

Generics cung cấp tính an toàn kiểu chỉ vào thời điểm dịch. Nếu bạn biên dịch tất cả mã của bạn với trình biên dịch Java 5.0 và không gặp bất kỳ cảnh báo chưa kiểm tra nào, việc kiểm tra vào thời điểm dịch đủ để đảm bảo mã của bạn cũng an toàn kiểu khi chạy. Nhưng nếu bạn có các cảnh báo không kiểm tra hay làm việc với legacy code thao tác các collections của bạn như các kiểu thô, bạn có thể muốn thực hiện các bước bổ xung để đảm bảo kiểu là an toàn vào thời điểm chạy. Bạn có thể làm điều này với các method như `checkedList()` và `checkedMap()` của `java.util.Collections`. Các method này kèm theo collection trong một collection bao (wrapper collection) thực hiện việc kiểm tra kiểu thời gian chạy để đảm bảo chỉ các giá trị đúng mới được thêm vào collection.

7.5.2 Một số lưu ý khi sử dụng Generics:

Không thể gọi Generics bằng dữ liệu Primitive

Ví dụ :

Có lớp generics `Pair<K,V>`;

Chúng ta không thể gọi `Pair<int, char>`;

Thay vào đó `Pair<Integer, Character>`;

Không thể tạo instances của kiểu dữ liệu Generics

Ví dụ:

E là dữ liệu Generics trong `List<E>`;

Không thể sử dụng `E tmp = new E()`;

Thay vào đó sử dụng `class<E>`;

Không thể sử dụng static cho Generics

```
public class Cat<T>{  
    private static T os;  
    //.....
```

```
};
```

Không thể ép kiểu hoặc sử dụng instanceof Ví

dụ:

```
public static <E> void it (List<E> list){ if(list  
    instanceof ArrayList<Integer>){  
    //...  
};
```

Hay:

```
ArrayList<Integer> list1 = new ArrayList<>();  
ArrayList<Number> list2 = (ArrayList<Number>) list1;
```

Không thể tạo mảng với parameterized types

```
List<Integer> [] list = new List<Integer>[2];
```

Không thể tạo ,catch , throw đối tượng parameterized types.

Không thể overload các hàm trong một lớp giống.

CHƯƠNG 8

THƯ VIỆN CÁC COLLECTION TRONG JAVA VÀ ÁP DỤNG.

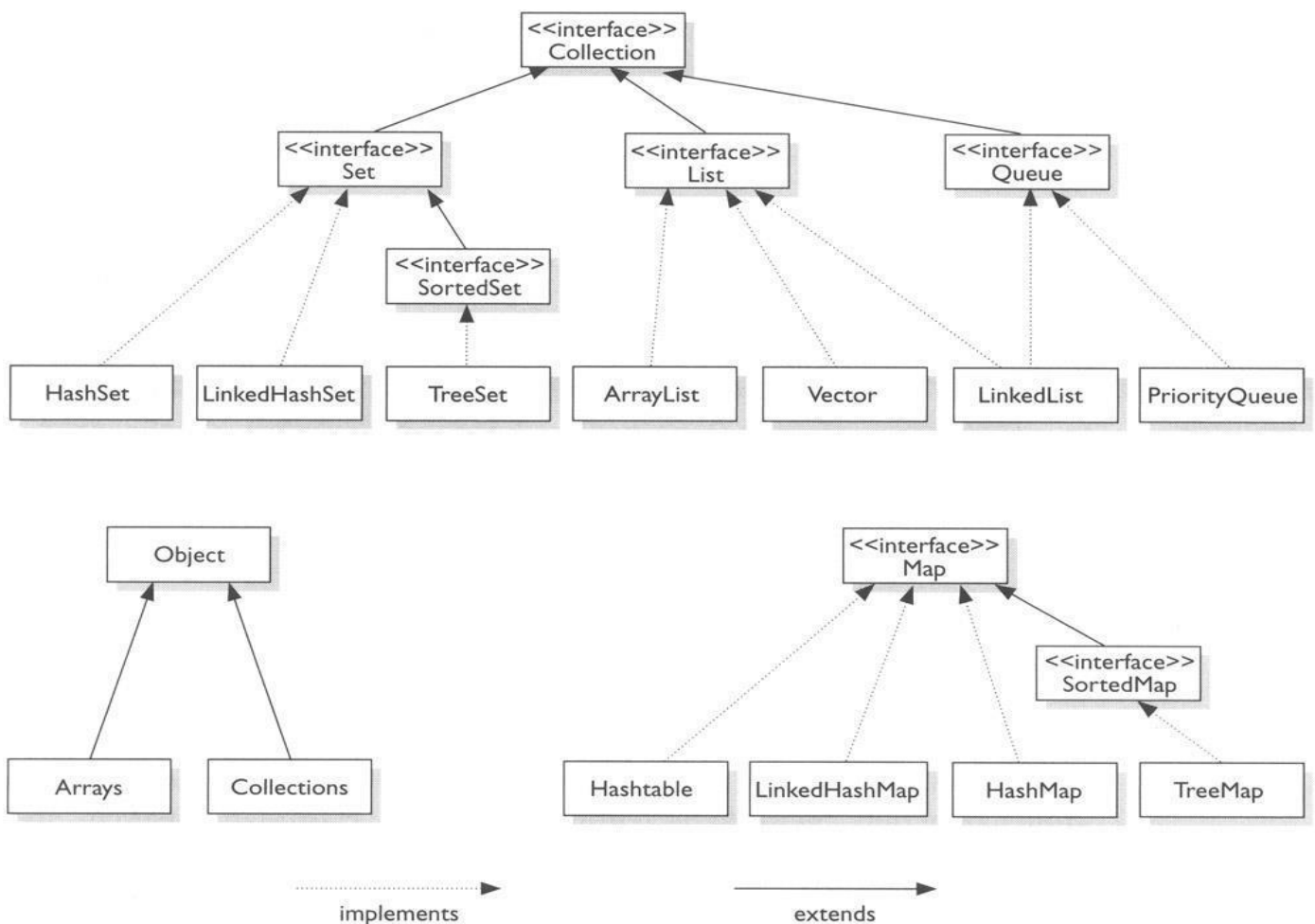
Chương này giới thiệu thư viện Collections, các thành phần, cách sử dụng. Đây là nội dung quan trọng để sinh viên nâng cao kỹ năng lập trình hướng đối tượng với Java

8.1 Các thành phần của Collection

Hình vẽ dưới đây mô tả cấu trúc chung của các thành phần trong thư viện collection nói chung. Trong đó:

Phía trên là các interface mô tả các kiểu cơ bản và được hiện thực hóa trong các lớp ở mức dưới.

Các lớp trong cấu trúc cây sẽ có tập hành động tương tự nhau dựa trên việc cài đặt các hàm thống nhất (giới thiệu trong phần sau).



Hình 8.1: Cấu trúc các lớp trong thư viện Collection

8.2 Giới thiệu Collection:

Định nghĩa: Một Collection đại diện cho 1 nhóm các đối tượng và được xác định như các phần tử của nó.

Collection interface được dùng để truyền tập các đối tượng mà yêu cầu tổng quát được đặt lên hàng đầu. Ví dụ, thông thường tất cả những cài đặt collection chung đều có 1 hàm khởi tạo sử dụng tham số kiểu Collection. Hàm này còn được gọi là hàm khởi tạo chuyển đổi (conversion constructor), giúp khởi tạo collection mới để có thể chứa được tất cả các phần tử có trong một Collection nào đó. Hay hiểu theo một cách khác, nó cho phép bạn chuyển kiểu Collection.

Giả sử, cho ví dụ, bạn có 1 Collection<String> c, có thể là List, Set hoặc 1 dạng nào đó của Collection. Khai báo này có thể tạo ra 1 ArrayList mới (1 cài đặt của interface List).

Khai báo: List<String> list = new ArrayList<String>(c);

Ví dụ về 1 số phương thức của Collection:

```
public interface Collection<E> extends Iterable<E> { //
    Thao tác cơ bản
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    // Tùy chọn
    boolean add(E element);
    // Tùy chọn
    boolean remove(Object element);
    Iterator<E> iterator();

    // Phép toán số lượng lớn (Bulk operations)
    boolean containsAll(Collection<?> c);
    // Tùy chọn
    boolean addAll(Collection<? extends E> c); //
    Tùy chọn
```

```

    boolean removeAll(Collection<?> c);
    // Tùy chọn
    boolean retainAll(Collection<?> c);
    // Tùy chọn
    void clear();

    // Thao tác trên mảng
    Object[] toArray();
    <T> T[] toArray(T[] a);
}

```

Qua những ví dụ trên, ta thấy được rằng Collection đã đại diện cho 1 nhóm các đối tượng như thế nào. Nó chứa những phương thức cho biết có bao nhiêu phần tử trong Collection (isEmpty, size), kiểm tra xem 1 phần tử nào đó có ở trong Collection hay không (contains), thêm, xóa phần tử (add, remove) và cung cấp vòng lặp trong toàn bộ Collection (iterator).

Phương thức add được định nghĩa một cách khá tổng quát. Phương thức sẽ bảo đảm rằng Collection sẽ chứa phần tử thêm vào sau khi lời gọi hoàn tất, và trả về true nếu Collection thay đổi theo lời gọi. Tương tự, phương thức remove được thiết kế để xóa một thể hiện của phần tử nhất định khỏi Collection, giả định rằng Collection có chứa phần tử, thì phương thức trả về true nếu Collection thay đổi theo lời gọi.

8.3 Duyệt Collection:

Có 2 cách duyệt Collection: (1) dùng cấu trúc for- each; (2) sử dụng Iterators **8.3.1**

Sử dụng for – each:

Cấu trúc for-each cho phép duyệt Collection hoặc Mảng sử dụng vòng for. Ví dụ sau in ra mỗi phần tử trên 1 dòng.

VD: *for (Object o : collection)*
System.out.println(o);

8.3.2 Sử dụng Iterator:

Iterator là 1 đối tượng cho phép duyệt Collection, xóa phần tử trong Collection 1 cách có chọn lựa. Tạo 1 đối tượng Iterator bằng cách gọi phương thức iterator. Dưới đây là interface Iterator:

```

public interface Iterator<E> {
    boolean hasNext();

```

```

    E next();
    void remove(); //optional
}

```

Phương thức hasNext trả về true nếu còn phần tử, và phương thức next trả về phần tử tiếp theo của vòng lặp. Phương thức remove xóa phần tử được trả về bởi next. Mỗi lần gọi hàm next chỉ được gọi 1 lần hàm remove, và exception sẽ sinh ra nếu quy tắc này không được tuân thủ.

Chú ý rằng Iterator.remove là cách an toàn DUY NHẤT để sửa Collection trong quá trình duyệt. Các hành vi không xác định có thể sinh ra nếu dùng cách khác để duyệt và sửa(ví dụ dùng cấu trúc for -each)

Sử dụng Iterator thay vì cấu trúc for – each trong các trường hợp:

Xóa phần tử hiện tại. Cấu trúc for- each ẩn vòng lặp đi, vì thế không thể gọi hàm remove. Do đó, cấu trúc for-each không sử dụng để lọc được.

Lặp song song qua nhiều Collection

Phương thức sau đây chỉ ra cách sử dụng 1 Iterator để lọc 1 Collection tùy ý (duyệt qua Collection và xóa phần tử chỉ định)

```

static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator(); it.hasNext(); )
        if (!cond(it.next()))
            it.remove();
}

```

Đoạn code trên mang tính đa hình, nó có thể thực hiện trên bất cứ Collection nào mà không quan tâm đến cách cài đặt. Ví dụ trên đã minh họa sự dễ dàng khi viết thuật toán đa hình sử dụng Java Collection Framework.

8.2.3 Thao tác số lượng lớn (Bulk)

Thao tác Bulk thực hiện 1 thao tác trên toàn bộ Collection. Có thể cài đặt những thao tác Bulk bằng những thao tác cơ bản, mặc dù trong phần lớn trường hợp, những cài đặt này tỏ ra kém hiệu quả hơn. Dưới đây là những thao tác Bulk:

containsAll – trả về true nếu Collection đích chứa toàn bộ các phần tử có trong Collection chỉ định.

addAll – thêm tất cả phần tử trong Collection chỉ định vào Collection đích.

removeAll – xóa tất cả những phần tử trong Collection đích mà CÓ TRONG Collection chỉ định.

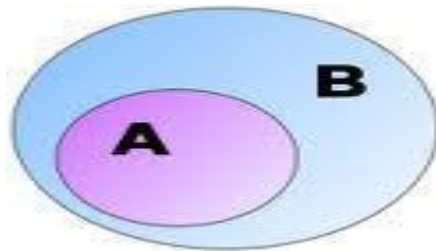
retainAll - xóa tất cả những phần tử trong Collection đích mà KHÔNG CÓ TRONG Collection chỉ định. Nghĩa là, nó chỉ giữ lại những phần tử có ở trong Collection chỉ định.

clear – xóa toàn bộ phần tử trong Collection.

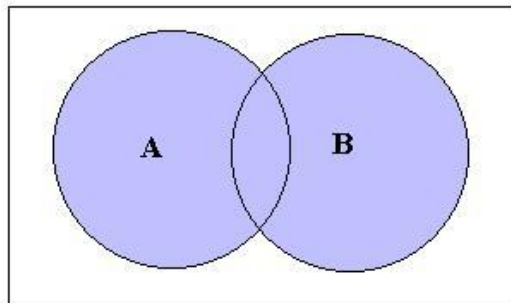
Giải thích:

Từ các định nghĩa trên, ta có thể mô tả được một số phương thức dựa trên lý thuyết tập hợp như sau:

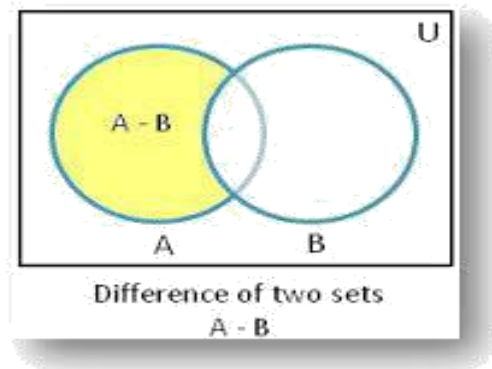
- o containsAll: Nhìn chung phương thức mô phỏng khá đúng mối quan hệ tập con – cha: từ hình trên ta thấy : Mọi phần tử của A phải nằm trong B hay $B.containsAll(A) == true$;



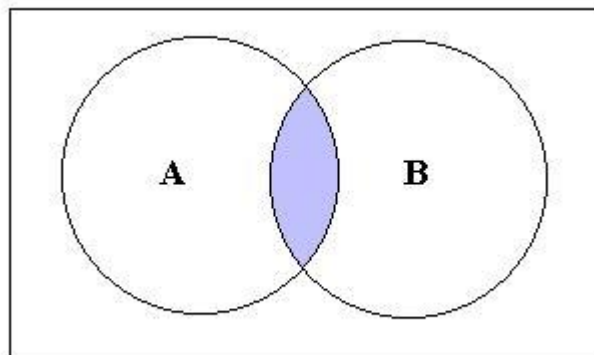
addAll: Là mô phỏng của phép hợp, ta cũng có thể biểu diễn dưới dạng giản đồ Venn:



removeAll: Dễ dàng nhận thấy đây là phép trừ 2 tập hợp: xóa những phần tử trong Collection hiện tại mà có trong Collection tham số. Ta chỉ lấy phần còn lại của Collection hiện tại



retainAll: Phương thức này là 1 mô phỏng của phép giao: Xóa phần tử trong A mà không có trong B. Vậy phần còn lại của tập A chính là giao của 2 tập.



Phương thức `addAll`, `removeAll`, `retainAll` trả về `true` nếu Collection đích được thay đổi trong quá trình thực hiện lời gọi thao tác.

Minh họa về khả năng của thao tác Bulk, ta lấy ví dụ về xóa tất cả các thể hiện của 1 phần tử chỉ định nào đó, phần tử `e`, khỏi Collection `c`.

```
c.removeAll(Collections.singleton(e));
```

Cụ thể hơn, giả sử ta muốn xóa toàn bộ những phần tử null khỏi Collection.

```
c.removeAll(Collections.singleton(null));
```

Câu lệnh sử dụng `Collections.singleton`, 1 phương thức tĩnh, trả về 1 Set bất biến chỉ chứa những phần tử xác định (khác null).

8.2.4 Thao tác mảng (array):

Phương thức `toArray` được sử dụng như 1 cầu nối giữa Collection và các API đời cũ (giao diện lập trình ứng dụng), cái muốn đầu vào (input) là mảng. Thao tác mảng cho phép nội dung của Collection có thể được chuyển thành 1 mảng.

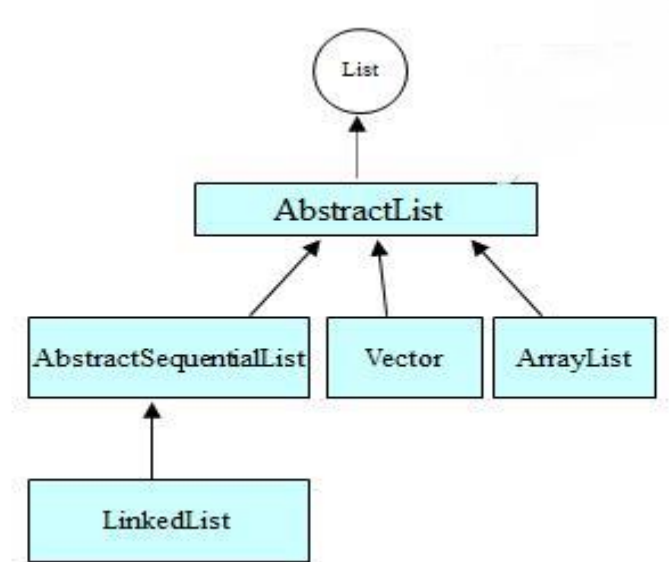
Ví dụ, giả sử `c` là 1 Collection. Đoạn code dưới đây đưa ra nội dung của `c` được biểu diễn bằng mảng Object, có độ dài mảng bằng với số phần tử trong `c`.

```
Object[] a = c.toArray();
```

Giả sử c chỉ chứa xâu (có thể c có kiểu `Collection<String>`). Đoạn code dưới đây đưa ra nội dung của c được biểu diễn bằng mảng String, độ dài mảng chính là số xâu có trong c.

```
String[] a = c.toArray(new String[0]);
```

8.3 Giới thiệu List và cách sử dụng



Định nghĩa: List là một Collection có thứ tự (hay còn gọi là 1 chuỗi) . List có thể chứa các thành phần giống nhau. Ngoài chức năng thừa hưởng từ Collection , List interface bao gồm:

Positional access (Truy cập vị trí) – quản lý các phần tử dựa trên vị trí của chúng trong List

Search (Tìm kiếm) - tìm kiếm một đối tượng quy định trong List và trả về số vị trí của nó.

Iteration (Lặp đi lặp lại) – vận dụng tính chất tuần tự của danh sách

Range-view - thực hiện các hoạt động phạm vi tùy ý trong danh sách.

Ví dụ List interface:

```
public interface List<E> extends Collection<E> {
```

Positional access

```
E get(int index);
```

optional

```
E set(int index, E element);
```

```

// optional
boolean add(E element);
// optional
void add(int index, E element);
// optional
E remove(int index);
// optional
boolean addAll(int index, Collection<? extends E> c);

// Search
int indexOf(Object o);
int lastIndexOf(Object o);

// Iteration
ListIterator<E> listIterator();
ListIterator<E> listIterator(int index);

// Range-view
List<E> subList(int from, int to);
}

```

Trong Java, INTERFACE LIST, bao gồm 3 phần là ARRAY LIST, VECTOR VÀ LINKED LIST. ArrayList giúp việc thực hiện hoạt động tốt hơn, và LinkedList cung cấp hiệu suất tốt hơn trong những tình huống nhất định. Ngoài ra, Vector đã được trang bị thêm để thực hiện List .

8.3.1 So sánh với Vector

Nếu bạn đã sử dụng Vector , bạn đã quen với những điều cơ bản của List . (Tất nhiên, List là một interface, trong khi Vector là một “thực thi” cụ thể) .List sửa một số thiếu sót API nhỏ trong Vector . Việc sử dụng Vector, chẳng hạn như phương thức elementAt và setElementAt , giúp ta đặt tên rõ ràng hơn. Ta xem ví dụ dưới đây:

```
a[i] = a[j].times(a[k]);
```

Vector tương đương:

```
v.setElementAt(v.elementAt(j).times(v.elementAt(k)), i);
```

List tương đương là:

```
v.set(i, v.get(j).times(v.get(k)));
```

Bạn có thể đã nhận thấy rằng phương thức `set` đã thay thế phương thức trong `Vector` là `setElementAt`, đảo ngược thứ tự của các đối số để chúng phù hợp với hoạt động mảng tương ứng. Xem xét ví dụ sau:

```
gift[5] = "golden rings";
```

Vector tương đương là:

```
gift.setElementAt("golden rings", 5);
```

List tương đương là:

```
gift.set(5, "golden rings");
```

Vì lợi ích của tính nhất quán của, phương thức `add (int, E)`, thay thế `insertElementAt (Object, int)`, cũng đảo ngược thứ tự của các đối số.

Ba thao tác phạm vi trong `Vector` (`indexOf`, `lastIndexOf`, và `setSize`) đã được thay thế bởi `Range-view (subList)` mạnh mẽ và phù hợp hơn rất nhiều.

8.3.2 ArrayList

Định nghĩa: `ArrayList` là một phiên bản thông minh hơn của mảng. Thuộc không gian tên `System.Collection.ArrayList`, lớp `ArrayList` có những đặc điểm của `Collection` (tập hợp) hơn là mảng như:

Kích thước mảng cố định theo khai báo còn `ArrayList` có thể tự động giãn theo yêu cầu.

Nếu mảng cần định kích thước, gán trị thì `ArrayList` cung cấp các phương thức cho phép thêm, chèn, xóa một phần tử trong tập hợp.

- Các phần tử của mảng phải cùng một kiểu dữ liệu, còn các phần tử của `ArrayList` có kiểu chung là `Object`, nghĩa là có thể có các kiểu khác.

Cách truy xuất đến một phần tử của `ArrayList` cũng như cách truy xuất phần tử của mảng.

8.3.3 Danh sách liên kết (Linked List)

Định nghĩa: Danh sách liên kết (linked list) là một cấu trúc dữ liệu bao gồm một nhóm các nút (nodes) tạo thành một chuỗi. Thông thường mỗi nút gồm dữ liệu (data) ở nút đó và tham chiếu (reference) đến nút kế tiếp trong chuỗi.

Danh sách liên kết là một trong những cấu trúc dữ liệu đơn giản và phổ biến nhất.

Ưu điểm:

Cung cấp giải pháp để chứa cấu trúc dữ liệu tuyến tính.

Dễ dàng thêm hoặc xóa các phần tử trong danh sách mà không cần phải cấp phát hoặc tổ chức lại trật tự của mảng.

Cấp phát bộ nhớ động

Nhược điểm:

Một danh sách liên kết đơn giản không cho phép truy cập ngẫu nhiên dữ liệu.

Chính vì lí do trên mà một số phép tính như tìm phần tử cuối cùng, xóa phần tử ngẫu nhiên hay chèn thêm, tìm kiếm có thể phải duyệt tất cả các phần tử.

Ví dụ khai báo:

```
Struct LLnode {  
    DataType Data;  
    LLnode* next;  
};
```

Các phép toán:

Khai báo:

```
struct LLintNode {  
    int Data;  
    struct LLintNode* Next;  
};
```

Đếm số phần tử của Linked List:

Duyệt từng phần tử rồi đếm, cho đến khi nào gặp phần tử cuối.

```
int LengthLL(LLNode* head) {  
    int length = 0;  
    while (head != NULL) {  
        ++length;  
        head = head ->Next;  
    }  
    return length;  
}
```

```
}
```

Thêm một phần tử vào cuối linked list:

Nếu danh sách rỗng, thêm nút vào head.

Ngược lại, tìm phần tử cuối cùng của danh sách rồi thêm nút mới vào Next của nút cuối cùng đó:

```
void AddLast(LLNode** head, int data) {
```

```
LLNode** tmp = head;
```

```
LLNode* NewNode;
```

```
NewNode = (LLNode*) malloc(sizeof(LLNode));
```

```
NewNode->Data = data;
```

```
NewNode->Next = NULL;
```

```
if ((*tmp) == NULL) {
```

```
(*tmp) = NewNode;
```

```
} else {
```

```
while ((*tmp)->Next != NULL) {
```

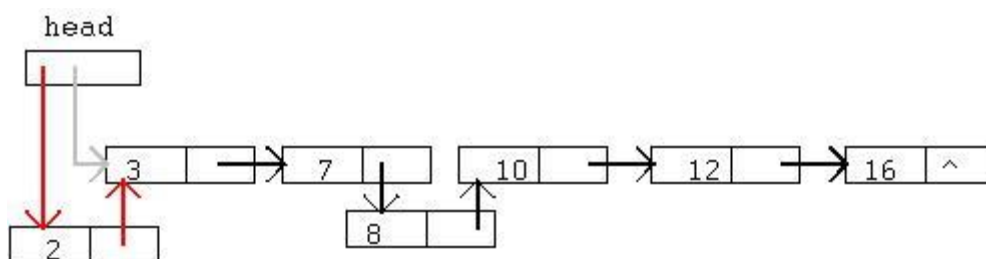
```
tmp = &((*tmp)->Next);
```

```
}
```

```
(*tmp)->Next = NewNode;
```

```
}
```

```
}
```



Ngoài những mô tả các thao tác kể trên, dựa vào cấu trúc trên ta hoàn toàn có thể tạo ra phương thức `addLast`, `removeFirst`, `insertAt...`

8.3.4 Giới thiệu Queue:

Định nghĩa: Một Queue là một Collection để lưu trữ các phần tử trước khi cài đặt việc truy cập. Bên cạnh các hoạt động Collection cơ bản, queue cung cấp việc chen thêm, loại bỏ, và các hoạt động kiểm tra. Interface `Queue` được cho sau đây:

```
public interface Queue<E> extends Collection<E> {
    E element();
    boolean offer(E e);
    E peek();
    E poll();
    E remove();
}
```

Mỗi phương thức Queue tồn tại dưới hai hình thức: (1) một ném một Exception (ngoại lệ) nếu hoạt động bị lỗi, và (2) ngoài ra là trả về một giá trị đặc biệt nếu hoạt động bị lỗi (hoặc null hoặc sai, tùy thuộc vào hoạt động). Cấu trúc thông thường của interface được minh họa trong bảng dưới đây.

CẤU TRÚC INTERFACE HÀNG ĐỢI

Loại hoạt động	Ném trả lại ngoại lệ	Trả lại giá trị đặc biệt
Insert	<code>add(e)</code>	<code>Offer(e)</code>
Remove	<code>remove()</code>	<code>Poll()</code>
Examine	<code>Element()</code>	<code>Peek()</code>

Queue thông thường, nhưng không nhất thiết, các phần tử được truy xuất theo thứ tự First In First Out (FIFO).. Trong số các Exception là `queue` ưu tiên, thứ tự phần tử tùy theo giá trị của nó. Bất cứ việc sắp thứ tự nào được sử dụng, phần tử đứng đầu của `queue` là các phần tử sẽ được gỡ bỏ bởi một lời gọi tới phương thức `remove` hoặc `poll`. Trong một `queue` FIFO, tất cả các phần tử mới được chen vào cuối hàng đợi. Các loại `queue` có thể sử dụng quy tắc vị trí khác nhau. Mỗi `queue` cài đặt phải xác định tính chất thứ tự của nó.

Có thể cài đặt để giới hạn số lượng phần tử trong queue ; queue như vậy được gọi là bị chặn. Một vài cài đặt Queue trong java.util.concurrent bị chặn, nhưng điều ngược lại với java.util.

Phương thức add, cái mà queue thừa kế từ Collection, là chèn một phần tử trừ khi nó sẽ vi phạm các hạn chế khả năng của queue, trong trường hợp nó ném `ExceptionIllegalStateException` .

Phương thức offer, được sử dụng trên Queue bị chặn, khác với add duy nhất ở chỗ trả về false nếu không thể thêm được phần tử vào.

Các phương thức remove và poll, cả hai đều xóa và trả lại (giá trị) đầu queue. Các phương thức remove và poll thể hiện sự khác nhau chỉ khi queue rỗng. Trong hoàn cảnh này, phương thức remove ném ra `NoSuchElementException`, trong khi poll trả về null.

Các phương thức peek và element trả về, nhưng không xóa phần tử đứng đầu của hàng đợi. Chúng khác nhau ở điểm: Nếu queue rỗng, phương thức element ném ra `Exception NoSuchElementException`, trong khi peek trả về null.

Queue thường không cho phép chèn các phần tử null. `LinkedList` được cài đặt từ Queue, là một ngoại lệ. Vì những lý do lịch sử, nó cho phép các phần tử null, nhưng bạn nên cẩn thận điều này, bởi vì null có thể là một giá trị trả lại đặc biệt khi sử dụng các phương thức poll và peek.

Interface Queue không định nghĩa các phương thức chặn queue, cái khá phổ biến trong lập trình song song. Những phương thức này, được định nghĩa trong gói interface `java.util.concurrent.BlockingQueue`.

Trong chương trình ví dụ sau đây, một queue được sử dụng để cài đặt một đồng hồ đếm ngược. Queue được tải trước vào với tất cả các giá trị số nguyên từ một số quy định đến 0, theo thứ tự giảm dần. Sau đó, các giá trị được lấy ra khỏi queue và in trong khoảng thời gian một giây:

```
import java.util.*;

public class Countdown {

    public static void main(String[] args) throws InterruptedException {

        int time = Integer.parseInt(args[0]);

        Queue<Integer> queue = new LinkedList<Integer>();

        for (int i = time; i >= 0; i--)

            queue.add(i);
```

```

        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
}

```

Trong ví dụ sau đây , một Queue ưu tiên (priority Queue) được sử dụng để SẮP XẾP một tập hợp các phần tử . Mục đích của đoạn chương trình là để kiểm tra sự sắp xếp của Queue ưu tiên:

Ví dụ:

```

static <E> List<E> heapSort(Collection<E> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    List<E> result = new ArrayList<E>();

    while (!queue.isEmpty())
        result.add(queue.remove());

    return result;
}

```

Interface Deque

Định nghĩa: Một deque là một double - ended – queue (hàng đợi 2 đầu). Một double-ended-queue là một tập hợp tuyến tính của các phần tử hỗ trợ chèn và loại bỏ các phần tử ở cả hai điểm đầu cuối. Interface Deque là một kiểu dữ liệu trừu tượng phong phú hơn cả Stack và Queue bởi vì nó cài đặt cả ngăn xếp và queue. Interface Deque , xác định phương thức để truy cập vào các phần tử ở cả hai đầu. Phương thức được cung cấp để chèn , xóa , và kiểm tra các phần tử. Các lớp tiền định nghĩa như ArrayDeque và LinkedList đều cài đặt interface Deque .

Lưu ý rằng interface Deque có thể được sử dụng như last-in- first-out Stack (LIFO) và first-in-first-out Queue (FIFO) . Các phương thức trong interface Deque được chia thành ba phần:

Thao tác chèn

Các phương thức `addFirst` và `offerFirst` chèn các phần tử vào đầu Deque . Các phương thức `addLast` và `offerLast` phần tử chèn vào cuối các ví dụ Deque . Khi Deque bị chặn, các phương thức ưa thích là `offerFirst` và `offerLast` vì `addFirst` hoặc `addLast` có thể không ném một Exception nếu Deque đầy .

Thao tác xóa

Các phương thức `removeFirst` và `pollFirst` loại bỏ các phần tử đầu ví dụ Deque . Các phương thức `removeLast` và `pollLast` loại bỏ các phần tử cuối cùng . Các phương thức `pollFirst` và `pollLast` trả lại null nếu Deque trống trong khi các phương thức `removeFirst` và `removeLast` ném một Exception trong trường hợp Deque trống .

Thao tác lấy phần tử

Các phương thức `getFirst` và `peekFirst` lấy phần tử đầu tiên của Deque. Những phương thức này không loại bỏ các giá trị từ Deque. Tương tự như vậy , các phương thức `getLast` và `peekLast` lấy phần tử cuối cùng nhưng không loại bỏ . Các phương thức `getFirst` và `getLast` ném một Exception trong trường hợp deque trống trong khi các phương thức `peekFirst` và `peekLast` trả lại NULL.

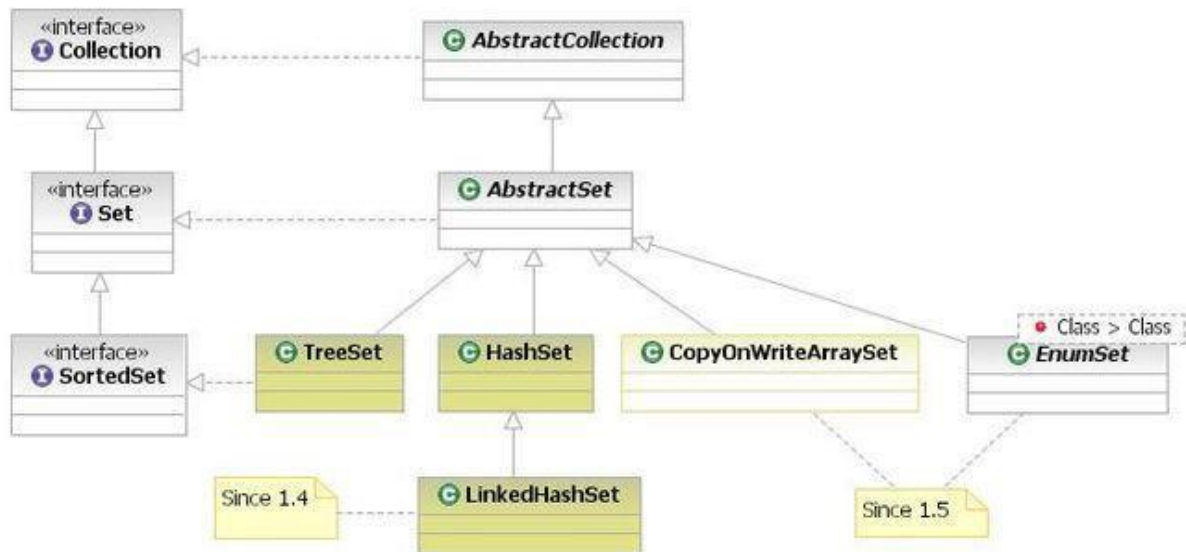
12 phương thức bao gồm chèn , loại bỏ và lấy phần tử được tóm tắt trong bảng dưới đây :

Type of Operation	First Element (Beginning of the Deque instance)	Last Element (End of the Deque instance)
Insert	<code>addFirst(e)</code> <code>offerFirst(e)</code>	<code>addLast(e)</code> <code>offerLast(e)</code>
Remove	<code>removeFirst()</code> <code>pollFirst()</code>	<code>removeLast()</code> <code>pollLast()</code>
Examine	<code>getFirst()</code> <code>peekFirst()</code>	<code>getLast()</code> <code>peekLast()</code>

Ngoài các phương thức cơ bản để chèn, xóa và kiểm tra một Deque, interface Deque cũng có một số phương thức được xác định trước. Một trong số đó là `removeFirstOccurrence` , phương thức này loại bỏ xuất hiện đầu tiên của các phần tử tham số nếu nó tồn tại trong Deque. Nếu phần tử không tồn tại sau đó các trường hợp

Deque vẫn không thay đổi . Một phương thức tương tự là `removeLastOccurrence` ; phương thức này loại bỏ sự xuất hiện cuối cùng của phần tử quy định nếu nó tồn tại trong Deque instance. Kiểu trả về của các phương thức này là boolean, và chúng trả về true nếu các phần tử tồn tại trong Deque.

8.4 Giới thiệu Set Interface:



Định nghĩa: Set là một loại Collections không chứa phần tử trùng nhau. Set biểu diễn một cách trừu tượng khái niệm tập hợp trong toán học. Trong interface Set chỉ bao gồm các phương thức được thừa kế từ Collections và thêm vào giới hạn là không cho phép có phần tử trùng nhau. Set cũng có những phương thức là equals và hashCode, cho phép những thể hiện của Set có thể dễ dàng so sánh với nhau ngay cả trong trường hợp chúng thuộc những loại thực thi khác nhau. Hai thể hiện được gọi là bằng nhau nếu chúng chứa những phần tử như nhau.

Dưới đây những phương thức nằm trong Set interface :

```
public interface Set<E> extends Collection<E> {
```

Toán tử cơ bản int

```
size(); boolean
```

```
isEmpty();
```

```
boolean contains(Object element);
```

Tùy chọn

```
boolean add(E element);
```

// Tùy chọn

```
boolean remove(Object element);
```

```
Iterator<E> iterator();
```

// Toán tử số lượng lớn

```
boolean containsAll(Collection<?> c);
```



```

// Tùy chọn
boolean addAll(Collection<? extends E> c);

// Tùy chọn
boolean removeAll(Collection<?> c);

// Tùy chọn
boolean retainAll(Collection<?> c);

// Tùy chọn
void clear();

    Toan tu cua mang
Object[] toArray(); <T>
T[] toArray(T[] a);
}

```

Nền tảng Java bao gồm 3 lớp thực thi chính của Set là HashSet, TreeSet, LinkedHashSet.

HashSet lưu trữ phần tử trong bảng băm (hash table). Đây là lớp thực thi cho tốc độ tốt nhất, tuy nhiên lại không đảm bảo thứ tự của phần tử.

TreeSet lưu trữ phần tử trong cây đỏ-đen (red – black tree), sắp xếp phần tử dựa trên giá trị của chúng, và do đó chậm hơn đáng kể so với HashSet.

LinkedHashSet cũng dùng bảng băm để lưu phần tử, ngoài ra còn sử dụng danh sách liên kết để sắp xếp phần tử theo thứ tự mà phần tử được chèn vào tập hợp (insertion-order).

LinkedHashSet giúp người dùng loại bỏ những trật tự hỗn độn không đoán trước gây ra bởi HashSet và với chi phí chỉ cao hơn một chút.

Dưới đây là một code đơn giản nhưng rất hay dùng với Set. Giả sử bạn có 1 đối tượng Collection và bạn muốn tạo một đối tượng Collection khác chứa những phần tử giống như đối tượng cũ nhưng loại bỏ những phần tử thừa

```
Collection<Type> noDups = new HashSet<Type>(c);
```

Đối tượng mới sẽ được loại bỏ những phần tử thừa vì nó thuộc loại Set, theo định nghĩa sẽ không được chứa phần tử trùng nhau.

Còn nếu muốn bảo toàn thứ tự phần tử như trong đối tượng gốc và cũng loại bỏ những phần tử bị trùng thì ta thay đoạn code trên bằng đoạn sau

```
Collection<Type> noDups = new LinkedHashSet<Type>(c);
```

Bên cạnh đó cũng có 1 phương thức generic bao trùm những đoạn code trên, trả về đối tượng Set với cùng kiểu mà chúng ta truyền vào . Đó là removeDups:

```
public static <E> Set<E> removeDups(Collection<E> c) {  
    return new LinkedHashSet<E>(c);  
}
```

Những toán tử cơ bản trong Set Interface (Basic Operations)

int size() : trả về số phần tử trong Set (lực lượng của chúng)

boolean isEmpty() : kiểm tra xem Set rỗng hay không , trả về true nếu Set rỗng

boolean add(E element) : thêm phần tử vào trong Set nếu nó chưa có trong Set đó, trả về true nếu chưa có phần tử trùng trong Set và false trong trường hợp ngược lại .

boolean remove(Object element) : xóa một phần tử được chỉ định trong Set. Trả về true nếu phần tử đó tồn tại trong Set và false trong trường hợp ngược lại.

Iterator<E> iterator() : trả về kiểu Iterator của Set

Chương trình dưới đây nhập vào danh sách đối số của hàm main và in ra những từ bị trùng, những từ khác nhau:

```
import java.util.*;
```

```
public class FindDups {  
    public static void main(String[] args) {  
        Set<String> s = new HashSet<String>();  
        for (String a : args)  
            if (!s.add(a))  
                System.out.println("Duplicate detected: " + a);  
  
        System.out.println(s.size() + " distinct words: " + s);  
    }  
}
```

Input là : i came i saw i left

Sau khi chạy chương trình trên ta được :

Duplicate detected: i

Duplicate detected: i

4 distinct words: [i, left, saw, came]

Chú ý rằng code luôn luôn tham chiếu tới Collection bởi kiểu Interface (ở đây là Set) hơn là kiểu thực thi (ở đây là HashSet) . Đây là điều đặc biệt trong việc luyện tập Java bởi nó tạo cho bạn sự linh hoạt trong việc thay đổi kiểu thực thi. Khi đó bạn chỉ cần thay đổi hàm khởi tạo .

Kiểu thực thi của Set ở ví dụ trên thuộc kiểu HashSet. Đó là kiểu không bảo toàn thứ tự của phần tử trong Set. Nếu bạn muốn chương trình in ra danh sách từ theo thứ tự Alphabet, đơn giản bạn chỉ cần thay đổi kiểu thực thi của Set từ HashSet chuyển thành TreeSet.

```
Set<String> s = new TreeSet<String>();
```

Kết quả in ra sẽ là

Duplicate detected: i

Duplicate detected: i

4 distinct words: [came, i, left, saw]

Những toán tử số lượng lớn trong Set Interface (Bulk Operations)

Những toán tử Bulk đặc biệt thích hợp với Set. Khi được áp dụng nó sẽ thực hiện những phép toán của tập hợp trong toán học. Giả sử ta có 2 tập hợp s1 và s2 . Sau đây là những phương thức thực hiện các phép toán Bulk trên 2 tập hợp s1 và s2 :

- s1.containsAll(s2) : trả về giá trị true nếu s2 là tập con của s1 (s2 là tập con của s1 nếu s1 chứa tất cả phần tử trong s2)
- s1.addAll(s2) : trả về tập hợp là hợp của s1 và s2 (hợp là tập hợp chứa tất cả phần tử của 2 tập hợp)
- s1.retainAll(s2) : trả về tập hợp là giao của s1 và s2 (giao là tập hợp chỉ chứa phần tử giống nhau giữa 2 tập hợp)
- s1.removeAll(s2) : trả về tập hợp bù của s2 với s1 (bù là tập hợp chứa các phần tử có trong s1 nhưng không có trong s2)

Để thực hiện những phép toán hợp, giao , lấy phần bù của 2 tập hợp mà không làm thay đổi 2 tập hợp, trước khi gọi các toán tử Bulk ta phải sao chép một tập hợp sang một tập mới . Đây là ví dụ :

```
Set<Type> union = new HashSet<Type>(s1);
union.addAll(s2);
```

```
Set<Type> intersection = new HashSet<Type>(s1);
intersection.retainAll(s2);
```

```
Set<Type> difference = new HashSet<Type>(s1);
difference.removeAll(s2);
```

Bây giờ ta sẽ tạo ra một chương trình FindDups . Giả sử bạn muốn biết từ nào trong danh sách từ chỉ xuất hiện một lần và từ nào xuất hiện nhiều hơn một lần. Tuy nhiên bạn lại không muốn lặp lại việc in ra phần tử trùng như ví dụ trên. Do đó thuật toán của chúng ta là tạo ra 2 tập hợp, 1 tập hợp chứa danh sách từ và tập kia chỉ chứa những từ bị trùng. Những từ mà chỉ xuất hiện một lần là phần bù của 2 tập hợp . Đây là chương trình FindDups :

```
import java.util.*;

public class FindDups2 {
    public static void main(String[] args) {
        Set<String> uniques = new HashSet<String>();
        Set<String> dups    = new HashSet<String>();

        for (String a : args)
            if (!uniques.add(a))
                dups.add(a);

        // Destructive set-difference
        uniques.removeAll(dups);

        System.out.println("Unique words:      " + uniques);
        System.out.println("Duplicate words: " + dups);
    }
}
```

```
}  
}
```

Giải thích code:

Lệnh `uniques.add(a)` ngoài việc thêm phần tử vào tập hợp, nó còn trả về kết quả thực hiện. Phương thức trả về `true` nếu thêm được, `false` nếu ngược lại.

Sau khi tạo được 2 tập hợp (`uniques` và `dups`), ta lấy phần bù (`uniques - dups`), kết quả là cho ra những phần tử chỉ thuộc tập `uniques` chứ không thuộc `dups` (nói cách khác chỉ giữ lại những phần tử xuất hiện duy nhất 1 lần)

Dĩ nhiên những phần tử xuất hiện trên (hoặc bằng) 2 lần sẽ ở trong tập `dups`

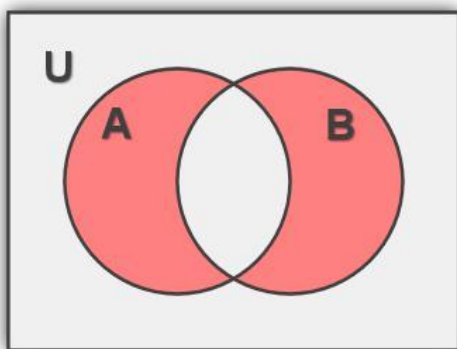
Khi chạy với danh sách đối số như ví dụ trước (`i came i saw i left`) kết quả sẽ là :

Unique words: [left, saw, came]

Duplicate words: [i]

Một phép toán ít phổ biến hơn là phép lấy phần bù cân xứng (symmetric set difference) – là tập hợp bao gồm phần tử của cả 2 tập hợp nhưng không chứa những phần tử giống nhau giữa 2 tập hợp. Đoạn code dưới đây sẽ thể hiện phép toán đó (Minh họa hình vẽ bên dưới):

```
Set<Type> symmetricDiff = new HashSet<Type>(s1);  
symmetricDiff.addAll(s2);  
Set<Type> tmp = new HashSet<Type>(s1);  
tmp.retainAll(s2);  
symmetricDiff.removeAll(tmp);
```



Giải thích code:

Đầu tiên ta “copy” nội dung của HashSet s1 vào SymmetricDiff

Sau đó tạo ra SymmetricDiff = s1 \cup s2 bằng phương thức addAll (đã đề cập kỹ trong phần Collection).

Tập tmp được tạo ra chính là s1 \cap s2

Bước cuối cùng là lấy phần bù: SymmetricDiff – tmp. Kết quả có thể minh họa như hình vẽ trên.

Toán tử với mảng trong Set Interface (Arrays Operations)

Toán tử với mảng trong Set không khác gì với những kiểu Collection khác . Nó đã được đề cập ở phần Collection Interface phía trên

Các kiểu thực thi của Set:

Các kiểu thực thi của Set bao gồm 2 loại là thực thi thông thường (general-purpose) và thực thi đặc biệt (special-purpose)

Kiểu thực thi thông thường (general-purpose)

Như đã giới thiệu ở phần trên, Set bao gồm 3 lớp thực thi thông thường đó là HashSet, TreeSet và LinkedHashSet.

Một điều đáng ghi nhớ là kiểu HashSet là một tập hợp tính tuyến tính giữa tổng số phần tử đưa vào với số lượng ô chứa (dung lượng) . Do đó lựa chọn dung lượng chứa khởi đầu quá lớn sẽ gây lãng phí không gian và thời gian. Mặt khác, lựa chọn dung lượng quá thấp sẽ làm tốn thời gian trong việc sao chép cấu trúc dữ liệu mỗi khi nó buộc phải tăng dung lượng lên. Nếu bạn không chỉ định dung lượng ban đầu, dung lượng mặc định sẽ là 16. Trong quá khứ, có một vài lợi thế khi lựa chọn dung lượng khởi đầu là những số nguyên tố. Tuy nhiên hiện nay điều này không còn đúng nữa. Thay vào đó sẽ là số mũ của 2. Dòng code dưới đây xác định một kiểu HashSet với dung lượng ban đầu là 64

```
Set<String> s = new HashSet<String>(64);
```

Kiểu thực thi đặc biệt (Special-purpose)

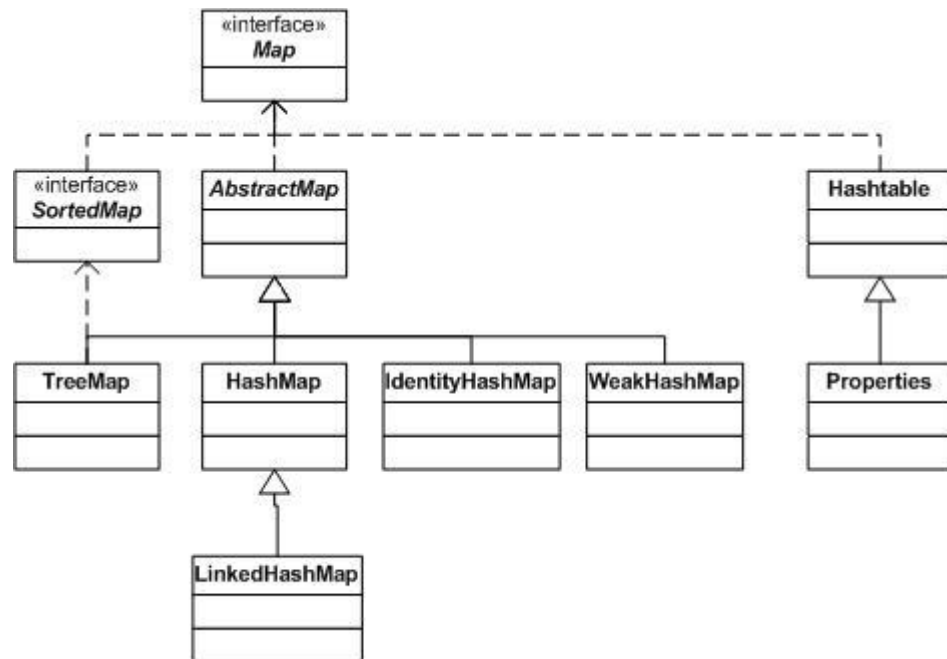
Set bao gồm 2 kiểu thực thi đặc biệt là EnumSet và CopyOnWriteArraySet.

EnumSet là kiểu thực thi hữu hiệu đối với những kiểu enum. Tất cả thành viên trong tập hợp enum phải cùng một kiểu enum. Tập hợp Enum hỗ trợ duyệt các phần tử trong giới hạn của nó. Ví dụ cho trước 1 enum là các ngày trong tuần, bạn có thể duyệt qua tất cả các ngày trong tuần như sau :

```
for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))
    System.out.println(d);
```

CopyOnWriteArraySet là kiểu thực thi theo phương pháp mảng copy-on-write. Tất cả những toán tử làm thay đổi tập hợp như add,set,and remove được thực hiện bằng cách tạo 1 bản sao của mảng. Không giống như hầu hết các kiểu thực thi của Set phương thức add ,remove và contains đòi hỏi thời gian cân xứng với kích thước của tập hợp. Do đó kiểu thực thi này chỉ thích hợp với những tập hợp mà chỉ cần duyệt phần tử chứ không cần thay đổi phần tử.

8.5 Giới thiệu về Map



Định nghĩa: Map là 1 đối tượng ánh xạ khóa tới giá trị. 1 Map không thể chứa khóa trùng nhau: mỗi khóa có thể ánh xạ đến nhiều nhất 1 giá trị. Map mô hình hóa khái niệm trừu tượng “hàm” trong Toán học. Dưới đây là interface Map:

```
public interface Map<K,V> {
```

```
// Thao tác cơ bản
```

```
V put(K key, V value);
```

```

V get(Object key);
V remove(Object key);
boolean containsKey(Object key);
boolean containsValue(Object value);
int size();
boolean isEmpty();

// Thao tác số lượng lớn
void putAll(Map<? extends K, ? extends V> m);
void clear();

```

```

Collection Views public
Set<K> keySet(); public
Collection<V> values();
public Set<Map.Entry<K,V>> entrySet();

```

```

Interface for entrySet elements
public interface Entry {
    getKey(); V
    getValue();
    V setValue(V value);
}
}

```

Có 3 cài đặt chính của Map trong nền tảng Java: HashMap, TreeMap và LinkedHashMap. Bổ sung bên cạnh là đối tượng Hashtable.

So sánh với Hashtable

Nếu đã sử dụng Hashtable, bạn đã quen với những khái niệm cơ bản của Map (mặc dù Map là interface trong khi Hashtable là implementation). Dưới đây là những khác biệt giữa Map và Hashtable:

Map cung cấp góc nhìn (views) thay cho việc hỗ trợ vòng lặp trực tiếp thông qua đối tượng Enumeration. Views giúp cải thiện đáng kể tính thể hiện của Interface

Map cho phép lập thông qua khóa(1), giá trị(2) hoặc cặp khóa – giá trị(3).
Hashtable không cho phép lựa chọn cuối cùng.

Map là 1 cách an toàn để xóa phần tử ở giữa vòng lặp, điều mà Hashtable không thể làm được.

Map đã sửa 1 vài thiếu sót của interface Hashtable. Cụ thể, Hashtable có phương thức contains, trả về true nếu Hashtable chứa giá trị cho trước. Interface Map sử dụng 2 hàm phân biệt là containsValue và containsKey, giúp cho việc xác định giá trị/khóa rõ ràng hơn và tránh được sự nhập nhằng.

Các thao tác cơ bản

Thao tác cơ bản của Map (put, get, containsKey, containsValue, size và isEmpty) có chức năng giống hệt các hàm này trong Hashtable. Chương trình dưới đây sinh ra kết quả là đếm số lần xuất hiện của các từ trong chuỗi cho trước.

Ví dụ:

```
import java.util.*;
```

```
public class Freq {  
    public static void main(String[] args) {  
        Map<String, Integer> m = new HashMap<String, Integer>();  
  
        Initialize frequency table from command line for  
        (String a : args) {  
            Integer freq = m.get(a);  
            m.put(a, (freq == null) ? 1 : freq + 1);  
        }  
  
        System.out.println(m.size() + " distinct words:");  
        System.out.println(m);  
    }  
}
```

Giải thích code: Điều đặc biệt trong đoạn code là tham số thứ 2 của phương thức put. Tham số này là biểu thức điều kiện có tác dụng đặt số lần xuất hiện là 1 nếu không tìm thấy từ trong Map và tăng lên 1 nếu từ đã có trong Map.

Input, output như bên dưới:

Input: java Freq if it is to be it is up to me to delegate

Output: 8 distinct words:

{to=3, delegate=1, be=1, it=2, up=1, if=1, me=1, is=2}

Nếu điều kiện đưa ra là in từ theo thứ tự bảng chữ cái, tất cả những gì ta cần làm chỉ là thay đổi cài đặt từ HashMap sang TreeMap.

Output: 8 distinct words:

{be=1, delegate=1, if=1, is=2, it=2, me=1, to=3, up=1}

Tương tự, nếu muốn in theo thứ tự “từ xuất hiện trước in trước” thì chỉ cần thay đổi cài đặt HashMap thành LinkedHashMap.

Output: 8 distinct words:

{if=1, it=2, is=2, to=3, be=1, up=1, me=1, delegate=1}

Cũng giống như interface Set và List, Map nói rộng yêu cầu với phương thức equals và hashCode, cho phép 2 đối tượng map có thể so sánh bằng nhau logic không quan tâm đến kiểu cài đặt (ví dụ có thể so sánh 2 đối tượng HashMap và LinkedHashMap). 2 thể hiện của Map là bằng nhau nếu chúng có cùng những ánh xạ khóa – giá trị.

Thông thường, tất cả những cài đặt chung của Map đều cung cấp hàm khởi tạo sử dụng 1 tham số Map cho phép khởi tạo 1 Map mới chứa toàn bộ cặp khóa – giá trị của Map tham số. Hàm khởi tạo chuyển đổi (conversion constructor) của Map hoàn toàn tương tự với (hàm khởi tạo chuyển đổi) của Collection: Cho phép người gọi tạo ra 1 Map với cách cài đặt mong muốn mà có chứa toàn bộ các ánh xạ trong một Map khác mà không quan tâm đến cách cài đặt của Map đó. Ví dụ, giả sử ta có 1 Map tên là m, Dòng code sau đây sẽ tạo ra một HashMap mới chứa toàn bộ cặp khóa – giá trị giống với m.

Map<K, V> copy = new HashMap<K, V>(m);

Các thao tác số lượng lớn (Bulk)

Thao tác clear giúp xóa toàn bộ các ánh xạ ra khỏi Map. Thao tác putAll của Map tương tự với thao tác addAll của Collection. Giả sử rằng Map được sử dụng để biểu diễn 1 tập các thuộc tính – giá trị, thao tác putAll, kết hợp cùng hàm khởi tạo chuyển đổi của Map, là 1 cách ngắn gọn để cài đặt thuộc tính với các giá trị mặc định. Đoạn code sau sẽ thể hiện điều đó:

```
static <K, V> Map<K, V> newAttributeMap(Map<K, V> defaults, Map<K, V> overrides) {  
    Map<K, V> result = new HashMap<K, V>(defaults);
```

```

    result.putAll(overrides);
    return result;
}

```

Góc nhìn Collection (Collection Views)

Các phương thức Collection View cho phép quan sát Map theo 3 cách sau:

`keySet` – Tập các khóa có trong Map

`values` – Tập hợp các giá trị có trong Map, có thể có nhiều khóa ánh xạ vào 1 giá trị.

`entrySet` – Tập các cặp khóa – giá trị chứa trong Map.

Collection views cung cấp cách để lặp trong Map. Ví dụ sau đây minh họa phương pháp lặp chuẩn (với các khóa) trong Map với cấu trúc `for – each`:

```
for (KeyType key : m.keySet())
```

```
    System.out.println(key);
```

Sử dụng iterator:

Filter a map based on some

property of its keys.

```
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
```

```
    if (it.next().isBogus())
```

```
        it.remove();
```

Đoạn code minh họa lặp theo cặp khóa – giá trị:

```
for (Map.Entry<KeyType, ValType> e : m.entrySet())
```

```
    System.out.println(e.getKey() + ": " + e.getValue());
```

Với góc nhìn của `entrySet`, cũng có thể thay đổi giá trị dựa trên khóa bằng cách gọi phương thức `Map.Entry setValue` trong quá trình lặp. Chú ý rằng `Iterator` là cách an toàn DUY NHẤT cho phép sửa Map trong quá trình lặp. Những sự kiện không xác định sẽ xảy ra nếu sửa Map theo một cách khác (vd: dùng cấu trúc `for - each`).

Collection views hỗ trợ xóa phần tử theo nhiều cách :`remove`, `removeAll`, `retainAll`, `clear`, `Iterator.remove` (giả sử là Map hỗ trợ xóa phần tử)

Collection views không hỗ trợ thêm phần tử trong mọi trường hợp. Thao tác thêm trở nên vô nghĩa trong `keySet` và `values` views, cũng như không cần thiết trong `entrySet` view, bởi vì chức năng `put` và `putAll` trong Map đã thực hiện thao tác thêm phần tử.

Cách sử dụng mới lạ của Collection Views (Map Algebra)

Multimaps

Một multimap giống như 1 Map nhưng mỗi khóa có thể ánh xạ đến nhiều hơn 1 giá trị. Java Collections Framework không bao gồm multimap interface do nó không được thường xuyên sử dụng. Cho ví dụ, đọc 1 danh sách từ, mỗi từ 1 dòng (tất cả chữ thường) và in ra những nhóm đảo chữ thỏa mãn kích cỡ. Một nhóm đảo chữ (anagram group) là 1 tập các từ, tất cả những chữ cái giống nhau nhưng thứ tự xuất hiện khác nhau. Chương trình sử dụng 2 tham số dòng lệnh (1) tên file input và (2) kích cỡ nhỏ nhất của nhóm đảo chữ in ra. Nhóm nào có ít từ hơn giá trị chỉ định sẽ không được in ra.

Có 1 số mẹo trong việc tìm nhóm đảo chữ: Với mỗi từ trong file input, sắp xếp nó lại theo thứ tự bảng chữ cái (alphabetize), đưa nó vào multimap và ánh xạ nó với từ nguyên gốc. Ví dụ, từ bad tạo ra sắp xếp cặp khóa – giá trị là <abd, bad> và đưa vào multimap. Nhiệm vụ sau cùng khá đơn giản, đó là duyệt qua multimap và đưa ra các nhóm thỏa mãn yêu cầu kích cỡ.

Chương trình sau được cài đặt thuần túy theo phương pháp trên:

```
import java.util.*;
```

```
import java.io.*;
```

```
public class Anagrams {
```

```
    public static void main(String[] args) {
```

```
        int minGroupSize = Integer.parseInt(args[1]);
```

```
        // Read words from file and put into a simulated multimap
```

```
        Map<String, List<String>> m = new HashMap<String, List<String>>();
```

```
        try {
```

```
            Scanner s = new Scanner(new File(args[0]));
```

```
            while (s.hasNext()) {
```

```
                String word = s.next();
```

```
                String alpha = alphabetize(word);
```

```
                List<String> l = m.get(alpha);
```

```
                if (l == null)
```

```
                    m.put(alpha, l=new ArrayList<String>());
```

```
                l.add(word);
```

```
            }
```

```

    } catch (IOException e) {
        System.err.println(e);
        System.exit(1);
    }

```

```

    Print all permutation groups above size threshold for
    (List<String> l : m.values())
        if (l.size() >= minGroupSize)
            System.out.println(l.size() + ": " + l);
}

```

```

private static String alphabetize(String s) {
    char[] a = s.toCharArray();
    Arrays.sort(a);
    return new String(a);
}
}

```

TÀI LIỆU THAM KHẢO

- [1]. *Advanced Java*. Aptech Worldwide Express.
- [2]. *Core Java*. Aptech Worldwide Express.
- [3]. Michael T. Goodrich and Roberto Tamassia, *Data Structures and Algorithms in Java*, 6th edition. . John Wiley & Sons Inc, 2014.
- [4]. Y, Daniel Liang, *Introduction to Java Programming*, 10th edition, 2014.
- [5]. H.M.Deitel and P.J. Deitel, *Java – How to Program*, 10th edition, Prentice Hall Inc, 2015.

PHỤ LỤC: BÀI TẬP LẬP TRÌNH JAVA

1. BÀI TẬP – JAVA CƠ BẢN

Bài 1. Nhập vào kích thước chiều rộng, chiều cao và in ra hình chữ nhật các dấu * nhưng rỗng bên trong. Ví dụ:

```
*****
*      *
*      *
*      *
*****
```

Bài 2. Nhập vào chiều cao và in ra hình như dưới đây (ví dụ với chiều cao bằng 4):

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
```

Bài 3. Viết chương trình nhập một số nguyên trong khoảng 10 .. 99. Sau đó lần lượt lấy ngẫu nhiên các số trong khoảng này cho đến khi có một giá trị đúng bằng giá trị nhập vào. In ra kết quả từng bước và số bước phải thực hiện.

Bài 4: Viết chương trình tính tổng $S = 1+3+5+ \dots + n$ nếu n lẻ

$S = 2 + 4 + \dots + n$ nếu n chẵn

Bài 5. Nhập số tự nhiên n rồi tính tổng (lưu ý phép chia các số nguyên):

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

Bài 6: Viết chương trình tìm **ước số chung lớn nhất** và bội số chung nhỏ nhất của hai số nguyên dương a,b.

Bài 7. Viết chương trình in ra các cặp số nguyên tố cùng nhau trong đoạn [a,b] với a,b nhập từ bàn phím.

Bài 8. Hãy viết chương trình tính tổng các chữ số của một số nguyên bất kỳ. Ví dụ: Số 8545604 có tổng các chữ số là: $8+5+4+5+6+0+4= 32$.

Bài 9. Viết chương trình phân tích một số nguyên thành các thừa số nguyên tố

Ví dụ: Số 28 được phân tích thành $2 \times 2 \times 7$

Bài 10. Viết chương trình liệt kê n số nguyên tố đầu tiên.

Bài 11. Dãy số Fibonacci được định nghĩa như sau: $F_0 = 0, F_1 = 1; F_i = F_{i-1} + F_{i-2}$. Nhập n, với $2 \leq n \leq 90$. Hãy viết chương trình in ra n số Fibonacci đầu tiên.

Bài 12. Viết chương trình liệt kê tất cả các số thuận nghịch có sáu chữ số.

Bài 13. Nhập một số $c > 0$ (ví dụ $c = 0.0001$) rồi tính số π theo công thức:

$$\pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + (-1)^n \frac{1}{2n+1})$$

tổng được tính với n đủ lớn sao cho bất đẳng thức $\frac{1}{2n+1} \leq c$ thỏa mãn.

Bài 14. Viết chương trình liệt kê các số thuận nghịch có 6 chữ số mà tổng chữ số chia hết cho 10 (ví dụ số: 721127).

Bài 15. Viết chương trình liệt kê các số nguyên có 7 đến 9 chữ số thỏa mãn:

- a) Là số thuận nghịch.
- b) Chỉ có chữ số 0, 6, 8
- c) Tổng chữ số chia hết cho 10

Bài 16. Viết chương trình liệt kê các số nguyên có 7 chữ số thỏa mãn:

- a) Là số nguyên tố
- b) Tất cả các chữ số là nguyên tố
- c) Đảo của nó cũng là một số nguyên tố

Bài 17. Nhập chiều cao (không quá 20) và xây dựng tam giác vuông chứa các số C_n^k theo công thức truy hồi sau:

$$C_n^0 = C_n^n = 1$$

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$$

Dòng thứ n của tam giác trên (bắt đầu từ $n=0$) chứa $n+1$ phần tử là các hệ số của nhị thức $(a+b)^n$ và được gọi là tam giác Pascal.

Bài 18. Tạo ngẫu nhiên dãy số a có n phần tử chỉ gồm các số nguyên dương có 3 chữ số, n không quá 50. Kiểm tra mảng a có phải là mảng đối xứng hay không (ví dụ: 15 2 1 2 15 là mảng đối xứng).

Bài 19. Tạo ngẫu nhiên dãy số a có không quá 1000 phần tử là các số nguyên dương có 3 chữ số. Đếm số lần xuất hiện của từng phần tử trong mảng. In phần tử có số lần xuất hiện nhiều nhất trong a .

Bài 20. Tạo ngẫu nhiên dãy số a có không quá 500 phần tử là các số nguyên dương có 4 chữ số. Nhập số X có 4 chữ số. Xác định vị trí của số nguyên tố trên a có giá trị gần với X nhất.

Bài 21. Tạo ngẫu nhiên mảng (a, N) gồm không quá 500 số nguyên dương có 3 chữ số.

- Sắp xếp dãy a tăng dần
- Loại bỏ bớt các phần tử trùng nhau sao cho mỗi giá trị chỉ xuất hiện trong a đúng một lần.

Bài 22. Nhập từ bàn phím 2 mảng (a, N) và (b, M) và số nguyên p ($0 \leq p < M < N < 50$). Hãy chèn mảng b vào vị trí p của a . Ví dụ: $(a, 4)$: **5 3 6 7**; $(b, 3)$: **2 9 11**; $p: 1 \rightarrow a, 7: 5$

2 9 11 3 6 7

Bài 23. Nhập dãy n số ($n \leq 1000$). Xác định đường chạy dài nhất, xuất lên màn hình vị trí phần tử đầu tiên và độ dài của đường chạy đó. Đường chạy là một dãy liên tiếp các phần tử không giảm của dãy ban đầu.

Ví dụ : Nhập dãy 1 4 2 3 1 2 6 8 3 5 7

Đường chạy dài nhất là : 4 4

Bài 24. Viết chương trình nhập số liệu cho ma trận các số nguyên A cấp $m \times n$ trong đó m, n là các số tự nhiên. Sau đó tìm ma trận chuyển vị $B = (b_{ij})$ cấp $n \times m$, $b_{ij} = a_{ji}$ $i = 1, 2, \dots, n; j = 1, 2, \dots, m$

Bài 25. Viết chương trình tính tích 2 ma trận các số nguyên A cấp $m \times n$ và B cấp $n \times k$.

Bài 26. Viết chương trình in ra ma trận xoáy ốc cấp $n \times n$ (với n không quá 50).

Bài 27. Sử dụng chuỗi ký tự để viết hàm kiểm tra số thuận nghịch. Áp dụng liệt kê các số thuận nghịch có 6 chữ số.

Bài 28. Tạo ngẫu nhiên hai số nguyên lớn (nhiều hơn 20 chữ số và không quá 1000 chữ số). Hãy tính tổng và hiệu hai số nguyên đó.

Bài 29. Tạo ngẫu nhiên hai số nguyên lớn (nhiều hơn 20 chữ số và không quá 1000 chữ số). Hãy tính tích của hai số nguyên đó.

Bài 30. Viết chương trình thực hiện chuẩn hoá một chuỗi ký tự nhập từ bàn phím (loại bỏ các dấu cách thừa, chuyển ký tự đầu mỗi từ thành chữ hoa, các ký tự khác thành chữ thường)

Bài 31. Viết chương trình thực hiện nhập một chuỗi ký tự và tìm từ dài nhất trong chuỗi đó. Từ đó xuất hiện ở vị trí nào? (Chú ý. nếu có nhiều từ có độ dài giống nhau thì chọn từ đầu tiên tìm thấy).

Bài 32. Viết chương trình thực hiện nhập một chuỗi họ tên theo cấu trúc: *họ...đệm...tên*; chuyển chuỗi đó sang biểu diễn theo cấu trúc *tên...họ...đệm*.

Bài 33. Hãy nhập và tính ra giá trị thập phân của một số viết dưới dạng LA MÃ.

Ví dụ: MDCLXVI = 1666. M:1000 ; D:500 ; C:100; L:50; X :10 ; V:5 ; I :1.

Bài 34. Nhập một câu không quá 20 từ, mỗi từ không quá 10 ký tự. Viết chương trình tách các từ trong câu và in các từ theo thứ tự Alphabet.

Bài 35. Nhập hai chuỗi ký tự $s1$ và $s2$ không quá 200 ký tự và chỉ bao gồm các chữ cái viết thường. Tìm hợp, giao, hiệu của hai tập các từ xuất hiện trong $s1$ và $s2$.

2. BÀI TẬP VIẾT CLASS

BÀI 1. Viết chương trình tạo lập lớp Sinh viên với các thuộc tính mã sinh viên, tên sinh viên, lớp, khóa. Khởi tạo một sinh viên với các giá trị nhập từ bàn phím. In thông tin sinh viên ra màn hình.

BÀI 2. Sử dụng lớp sinh viên trong bài 1, nhập vào danh sách n sinh viên và thực hiện:

- In toàn bộ danh sách ra màn hình
- Sắp xếp danh sách theo tên sinh viên
- In danh sách sinh viên theo lớp

BÀI 3. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng hình chữ nhật có tên là Rectangle theo lược đồ UML dưới đây:

Rectangle	
- width: double	Chiều rộng hình chữ nhật
- height: double	Chiều dài hình chữ nhật
- <u>color: String</u>	Màu hình chữ nhật
+ Rectangle()	Tạo HCN có c.dài = 1, c.rộng = 1
+ Rectangle(width: double, height: double, color: String)	Tạo HCN có c.dài, c.rộng xác định qua tham số
+ getWidth() :double	Trả về chiều rộng
+ setWidth(width:double): void	Thiết lập chiều rộng mới
+ getHeight(): double	Trả về chiều dài
+ setHeight(height:double): void	Thiết lập chiều dài mới
+ <u>getColor(): String</u>	Trả về màu của HCN
+ <u>setColor(color): void</u>	Thiết lập màu mới cho HCN
+ findArea(): double	Tính và trả về diện tích HCN
+ findPerimeter(): double	Tính và trả về chu vi HCN

- Từ chương trình chính, tạo một đối tượng hình chữ nhật với kích thước và màu sắc nhập vào từ bàn phím. Đưa ra màn hình các thuộc tính, diện tích và chu vi của đối tượng hình chữ nhật đã tạo.

BÀI 4. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng ngăn xếp chứa các số nguyên có tên là StackOfIntegers theo lược đồ UML dưới đây:

StackOfIntegers	
- elements: int[]	Mảng chứa các số nguyên trong stack
- size: int	Số lượng số nguyên đang chứa trong stack
+ StackOfIntegers()	Tạo một stack rỗng có dung lượng = 16
+ StackOfIntegers (capacity: int)	Tạo một stack rỗng có dung lượng xác định trong tham số
+ isEmpty(): boolean	Trả về true nếu stack rỗng
+ isFull(): boolean	Trả về true nếu stack đầy
+ peak(): int	Trả về số nguyên ở đỉnh stack
+ push(value:int): void	Đẩy một số nguyên vào đỉnh stack
+ pop(): int	Lấy ra và trả về số nguyên tại đỉnh stack
+ getSize(): int	Trả về số lượng số nguyên đang chứa trong stack

Viết chương trình sử dụng lớp StackOfIntegers để đưa ra màn hình tất cả các số nguyên tố nhỏ hơn một số nguyên dương n (nhập vào từ bàn phím) theo thứ tự giảm dần.

BÀI 5. Tạo lớp StackOfIntegers như bài 4. Viết chương trình nhập vào một số nguyên dương rồi hiển thị các thừa số nguyên tố nhỏ nhất của nó theo thứ tự ngược. Ví dụ, nếu số nguyên là 120, các thừa số nguyên tố nhỏ nhất được hiển thị là 5, 3, 2, 2, 2. Sử dụng lớp StackOfIntegers để chứa các thừa số rồi lấy và hiển thị chúng theo thứ tự ngược.

BÀI 6. Tạo lớp StackOfIntegers như bài 4, viết chương trình đổi một số nguyên bất kỳ sang cơ số 2.

BÀI 7. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng ngăn xếp chứa các ký tự có tên là StackOfChars theo lược đồ UML dưới đây:

StackOfChars	
- elements: char[]	Mảng chứa các ký tự trong stack
- size: int	Số lượng ký tự đang chứa trong stack
+ StackOfChars()	Tạo một stack rỗng có dung lượng = 16
+ StackOfChars (capacity: int)	Tạo một stack rỗng có dung lượng xác định trong tham số
+ isEmpty(): boolean	Trả về true nếu stack rỗng
+ isFull(): boolean	Trả về true nếu stack đầy
+ peak(): char	Trả về ký tự ở đỉnh stack
+ push(ch:char): void	Đẩy một ký tự vào đỉnh stack
+ pop(): char	Lấy ra và trả về ký tự tại đỉnh stack
+ getSize(): int	Trả về số lượng ký tự đang chứa trong stack

Viết chương trình sử dụng lớp StackOfChars để tính giá trị của biểu thức số học dạng trung tố có dấu ngoặc đầy đủ. Giả sử trong biểu thức số học chỉ chứa các phép toán cộng, trừ, nhân, chia và các số hạng là các số chỉ có một chữ số.

Ví dụ: Nhập vào biểu thức số học $((3+7) \times (9-(6-2)))$, đưa ra kết quả là 50.

BÀI 8. Tạo lớp StackOfChars như bài 7. Viết chương trình sử dụng lớp StackOfChars để chuyển một biểu thức dạng trung tố về dạng hậu tố.

BÀI 9. Tạo lớp StackOfChars như bài 7. Viết chương trình sử dụng lớp StackOfChars để đổi số nguyên n bất kỳ sang cơ số b bất kỳ ($1 < b < 36$).

BÀI 10. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng điểm trong mặt phẳng tọa độ OXY có tên là MyPoint theo lược đồ UML dưới đây:

MyPoint	
- x: double	Tọa độ x
- y: double	Tọa độ y
+ MyPoint()	Tạo đối tượng mặc định
+ MyPoint(x: double, y: double)	Tạo đối tượng có tọa độ xđ trong tham số
+ MyPoint(p: MyPoint)	Tạo đối tượng là bản sao của đối tượng trong t.số
+ getX(): double	Trả về tọa độ X
+ getY(): double	Trả về tọa độ Y
+ distance(secondPoint: MyPoint): double	Trả về khoảng cách từ điểm này tới điểm thứ hai
+ distance(p1: MyPoint, p2: MyPoint): double	Trả về khoảng cách giữa hai điểm

Viết chương trình tạo n đối tượng điểm MyPoint có tọa độ nhập vào từ bàn phím. Tìm hai điểm có khoảng cách lớn nhất. Đưa ra màn hình tọa độ của hai điểm tìm được và giá trị khoảng cách giữa chúng.

BÀI 11. Sử dụng lớp MyPoint ở câu trên và nhập vào 3 điểm trong không gian 2 chiều. Kiểm tra xem có tạo thành một tam giác hay không, có phải là tam giác dạng đặc biệt (vuông, cân, vuông cân, hay đều) hay không.

BÀI 12. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng ma trận có tên là Matrix theo lược đồ UML dưới đây:

Matrix	
- a: float[][] - rows: int - cols: int	Mảng chứa các phần tử của ma trận
+ Matrix()	Tạo ma trận có số hàng và số cột bằng 3
+ Matrix(rows: int, cols : int)	Tạo ma trận có số hàng và số cột xđ trong tham số
+ add(m: Matrix): Matrix	Tính tổng ma trận này với ma trận trong tham số, trả về ma trận tổng. Trong phương thức có kiểm tra sự hợp lệ về số hàng, số cột của hai ma trận.
+ sub(m: Matrix): Matrix	Tính hiệu ma trận này với ma trận trong tham số, trả về ma trận hiệu. Trong phương thức có kiểm tra sự hợp lệ về số hàng, số cột của hai ma trận.
+ neg(): Matrix	Đổi dấu các phần tử của ma trận, trả về ma trận đã đổi dấu các phần tử
+ transpose(): Matrix	Trả về ma trận chuyển vị
+ mul(m: Matrix): Matrix	Tính tích ma trận này với ma trận trong tham số, trả về ma trận tích. Trong phương thức có kiểm tra sự hợp lệ về số hàng, số cột của hai ma trận.
+ print(): void	Đưa ma trận ra màn hình
+ input(): void	Nhập vào các phần tử của ma trận

- Viết chương trình sử dụng lớp Maxtrix để thực hiện một số phép toán về ma trận.

Chương trình có các mục menu:

1. Tính tổng và hiệu hai ma trận;
2. Tính tích hai ma trận;
3. Tìm chuyển vị của một ma trận;
4. Kết thúc chương trình.

Khi người sử dụng chọn các mục từ 1 đến 3 thì cho nhập vào ma trận, thực hiện tính toán và đưa ra kết quả; khi người sử dụng chọn 4 thì kết thúc chương trình.

BÀI 13. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng phân số có tên là PhanSo theo lược đồ UML dưới đây:

PhanSo	
- ts: int	Tử số
- ms: int	Mẫu số
+ PhanSo()	Tạo phân số có tử số bằng 0, mẫu số bằng 1
+ PhanSo(ts: int, ms: int)	Tạo phân số có tử số và mẫu số xđ trong tham số
+ cong(sp2: PhanSo): PhanSo	Cộng hai phân số
+ tru(sp2: PhanSo): PhanSo	Trừ hai phân số
+ nhan(sp2: PhanSo): PhanSo	Nhân hai phân số
+ chia(sp2: PhanSo): PhanSo	Chia hai phân số
+ nghichDao(): PhanSo	Nghịch đảo phân số và trả về phân số nghịch đảo
+ doiDau(): PhanSo	Đổi dấu phân số và trả về phân số đã đổi dấu
+ toiGian(): PhanSo	Tối giản phân số và trả về phân số đã tối giản
+ bangNhuau(ps2: PhanSo): boolean	So sánh bằng
+ lonHon(ps2: PhanSo): boolean	So sánh lớn hơn
+ nhoHon(ps2: PhanSo): boolean	So sánh nhỏ hơn
+ hien(): void	Đưa phân số ra màn hình ở dạng ts/ms
+ nhap(): void	Nhập vào phân số dạng ts/ms

- Viết chương trình sử dụng lớp PhanSo. Nhập vào hai phân số; tính tổng, hiệu, tích, thương hai phân số; tối giản và so sánh hai phân số.

BÀI 14. Viết chương trình thực hiện các yêu cầu sau:

- Tạo lớp đối tượng số phức có tên là SoPhuc theo lược đồ UML dưới đây:

SoPhuc	Số phức $a + jb$
- a: float	Phần thực a
- b: float	Phần ảo b
+ SoPhuc()	Tạo số phức có phần thực và ảo bằng 0
+ SoPhuc(thuc: float, ao: float)	Tạo số phức có phần thực và ảo xđ trong tham số
+ cong(sp2: SoPhuc): SoPhuc	Cộng hai số phức
+ tru(sp2: SoPhuc): SoPhuc	Trừ hai số phức
+ nhan(sp2: SoPhuc): SoPhuc	Nhân hai số phức
+ chia(sp2: SoPhuc): SoPhuc	Chia hai số phức
+ nghichDao(): SoPhuc	Trả về số phức nghịch đảo
+ bangNau(sp2: SoPhuc): boolean	So sánh bằng
+ lonHon(sp2: SoPhuc): boolean	So sánh lớn hơn (theo modul)
+ nhoHon(sp2: SoPhuc): boolean	So sánh nhỏ hơn (theo modul)
+ hien(): void	Đưa số phức ra màn hình ở dạng $a + jb$
+ nhap(): void	Nhập vào số phức dạng $a + jb$

Viết chương trình sử dụng lớp SoPhuc. Nhập vào hai số phức; tính tổng, hiệu, tích, thương hai số phức; tính nghịch đảo và so sánh hai số phức.

BÀI 15. Cho lớp Sinh viên được mô tả như sau:

SinhVien	
- maSV: int	Mã SV: có 4 chữ số
- ten: String	Tên sinh viên, gồm cả họ và tên. Không được để trống
- lop: String	Lớp: Theo quy tắc đặt tên của PTIT.
- dtb: double	Điểm trung bình: từ 0 đến 10.
+ SinhVien()	Khởi tạo đối tượng mặc định
+ SinhVien(m: int, t: String, l: String, d: double)	Khởi tạo đối tượng có tham số
.....	(... cần bổ sung các phương thức cho phù hợp với yêu cầu ...)

Hãy thực hiện các yêu cầu sau:

1. Cho file dữ liệu SV.INP trong đó mỗi sinh viên ghi trên 4 dòng lần lượt là mã, tên, lớp, điểm. Nhập dữ liệu và lưu vào mảng. Chú ý kiểm tra các ngoại lệ theo ràng buộc của các thuộc tính, nếu sai bỏ qua sinh viên đó (tức là chỉ nhận các sinh viên có thông tin đúng theo ràng buộc).
2. In danh sách sinh viên đã nhập ra màn hình. Mỗi sinh viên ghi trên một dòng, mỗi thông tin cách nhau đúng một khoảng trống.
3. Sắp xếp danh sách sinh viên theo lớp và hiển thị danh sách sinh viên trong từng lớp theo điểm trung bình giảm dần. Ghi ra file SX.OUT. Mỗi sinh viên ghi trên 1 dòng, các thông tin cách nhau đúng một khoảng trống.
4. Phân loại sinh viên theo 4 mức: Giỏi ($đtb > 8$), Khá ($7 < đtb < 8$), Trung Bình ($5 < đtb < 7$), Yếu ($đtb < 5$). Sắp xếp danh sách đã phân loại theo tên sinh viên (tách tên và sắp xếp theo ABC, nếu trùng tên thì sắp xếp theo họ và tên đệm). Ghi ra file XEPLOAI.OUT.

Chú ý: Sinh viên cần bổ sung các lớp cho phù hợp.

BÀI 16. Cho lớp Mặt Hàng được mô tả như sau:

MatHang	
- maHang: int	Mã hàng: có 4 chữ số
- ten: String	Tên hàng, không được để trống
- nhom: String	Nhóm: gồm: hàng tiêu dùng, hàng thời trang, điện tử - điện lạnh
- giaBan: double	Giá bán: lớn hơn 0.
+ MatHang()	Khởi tạo đối tượng mặc định
+ MatHang(m: int, t: String, nhom: String, gia: double)	Khởi tạo đối tượng có tham số
.....	(... cần bổ sung các phương thức cho phù hợp với yêu cầu ...)

Hãy thực hiện các yêu cầu sau:

1. Cho file MH.INP trong đó mỗi mặt hàng ghi trên 4 dòng: mã, tên, nhóm, giá bán. Nhập dữ liệu và lưu vào mảng. Chú ý kiểm tra các ngoại lệ theo ràng buộc của các thuộc tính, nếu sai yêu bỏ qua mặt hàng đó.
2. In danh sách mặt hàng đã nhập ra màn hình. Mỗi mặt hàng ghi trên một dòng, mỗi thông tin cách nhau đúng một khoảng trống.
3. Sắp xếp và ghi danh sách mặt hàng theo giá bán giảm dần vào file SX.OUT. Mỗi mặt hàng ghi trên một dòng, mỗi thông tin cách nhau 1 khoảng trống.
4. Ghi ra danh sách mặt hàng theo từng nhóm, trong mỗi nhóm sắp xếp theo tên hàng theo thứ tự ABC. Ghi vào file NHOM.OUT.

Chú ý: Sinh viên cần bổ sung các lớp cho phù hợp.

BÀI 17. Cho lớp Sách được mô tả như sau:

Sach	
- maSach: int	Mã sách: có 4 chữ số
- tenSach: String	Tên sách, không được để trống
- chuyenNganh: String	Chuyên ngành: gồm: Giáo khoa, Văn học, Kỹ thuật,
- giaBan: double	Giá bán: lớn hơn 0.
+ Sach()	Khởi tạo đối tượng mặc định
+ Sach(m: int, t: String, cn : String, gia: double)	Khởi tạo đối tượng có tham số
.....	(... cần bổ sung các phương thức cho phù hợp với yêu cầu ...)

Hãy thực hiện các yêu cầu sau:

1. Cho file dữ liệu SACH.INP trong đó mỗi cuốn sách ghi trên 4 dòng gồm: mã sách, tên sách, chuyên ngành, giá bán. Nhập dữ liệu và lưu vào mảng. Chú ý kiểm tra các ngoại lệ theo ràng buộc của các thuộc tính, nếu sai bỏ qua cuốn sách đó.
2. In danh sách các cuốn sách ra màn hình, mỗi cuốn sách trên một dòng, mỗi thông tin cách nhau đúng một khoảng trống.
3. Sắp xếp danh sách các cuốn sách theo giá bán giảm dần và ghi vào file SX.OUT, mỗi cuốn sách ghi trên một dòng, mỗi dòng cách nhau một khoảng trống.
4. Ghi ra danh sách các cuốn sách theo từng chuyên ngành, trong mỗi chuyên ngành sắp xếp tên sách theo thứ tự ABC. Lưu vào file CN.OUT.

Chú ý: Sinh viên cần bổ sung các lớp cho phù hợp.

BÀI 18. Cho lớp Môn Học được mô tả như sau:

MonHoc	
- maMon: int	Mã môn: có 4 chữ số
- tenMon: String	Tên môn, không được để trống
- nhomMon: String	Nhóm môn: gồm: Cơ bản, cơ sở ngành, chuyên ngành, kỹ năng mềm.
- soTinchi: int	Số tín chỉ: lớn hơn 0, nhỏ hơn 7.
+ MonHoc()	Khởi tạo đối tượng mặc định
+ MonHoc(m: int, t: String, nhom :String, so: int)	Khởi tạo đối tượng có tham số
.....	(... cần bổ sung các phương thức cho phù hợp với yêu cầu ...)

Hãy thực hiện các yêu cầu sau:

1. Cho file dữ liệu MH.INP trong đó mỗi môn học ghi trên 4 dòng gồm mã môn, tên môn, nhóm môn, số tín chỉ. Nhập thông tin và lưu vào mảng. Chú ý kiểm tra các ngoại lệ theo ràng buộc của các thuộc tính, nếu sai thì bỏ qua môn học đó.
2. In danh sách môn học ra màn hình, mỗi môn học trên một dòng, mỗi thông tin cách nhau đúng một khoảng trống.
3. Sắp xếp danh sách môn học theo số tín chỉ giảm dần và ghi vào file SX.OUT.
4. Ghi ra danh sách các môn học theo từng nhóm môn. Trong mỗi nhóm sắp xếp tên môn theo ABC. Lưu vào file NHOM.OUT.

Chú ý: Sinh viên cần bổ sung các lớp cho phù hợp.

3. BÀI TẬP LẬP TRÌNH GIAO DIỆN VÀ XỬ LÝ SỰ KIỆN

Yêu cầu chung: Sinh viên **KHÔNG** sử dụng kéo thả.

- BÀI 1.** Viết chương trình tạo một hộp chọn màu. Khi chọn đến màu nào thì màu màn hình chuyển sang màu đó (có thể chọn màu dạng List, ComboBox hoặc ColorChooser)
- BÀI 2.** Viết chương trình vẽ các hình cơ bản: hình vuông, hình chữ nhật, hình tròn, hình đa giác. Cho phép người dùng chọn loại hình vẽ, kiểu rỗng hay đặc và chọn màu của hình. Cải tiến chương trình để có thể thay đổi các tham số của từng hình khi vẽ.
- BÀI 3.** Viết chương trình vẽ lên giao diện một đồng hồ có ba kim giờ, phút, giây và các số 3, 6, 9, 12. Ở phía dưới, hiển thị thêm một đồng hồ khác với giá trị số *giờ:phút:giây* (định dạng 24h).
- BÀI 4.** Viết chương trình mô phỏng máy tính bỏ túi với các chức năng cơ bản: cộng, trừ, nhân chia, lũy thừa, căn bậc hai.
- BÀI 5.** Viết chương trình cho phép chọn file ảnh trong ổ cứng và hiển thị lên giao diện. Cho phép thay đổi kích thước khung hiển thị và hiển thị nhiều ảnh.
- BÀI 6.** Tạo giao diện cho ít nhất 5 bài trong danh sách các bài tập cơ bản làm quen với Java (bài tập số 1). Chú ý bắt lỗi người dùng khi nhập.
- BÀI 7.** Tạo giao diện cho các bài tập viết class (bài tập số 2). Trong đó có nút nhấn cho phép đọc thông tin từ file và hiện ra trên Bảng (JTable), nút sắp xếp cho phép hiện ra danh sách đã sắp xếp.

4. BÀI TẬP VỀ QUAN HỆ GIỮA CÁC CLASS

BÀI 1: QUẢN LÝ BÁN HÀNG TRONG SIÊU THỊ

Thông tin về **Mặt hàng** gồm các thuộc tính (*mã hàng, Tên hàng, Nhóm hàng, Giá bán*) – trong đó Nhóm hàng có thể là: Hàng thời trang, Hàng tiêu dùng, Hàng điện máy, Hàng gia dụng. Mã hàng là một số nguyên có 4 chữ số.

Thông tin về **Khách hàng** (*Mã KH, Họ tên, Địa chỉ, Số ĐT*), mã KH là một số nguyên dương có 4 chữ số; số điện thoại có từ 9 đến 10 chữ số và bắt đầu bằng chữ số 0.

Một **Hóa đơn** sẽ có mã hóa đơn và thời gian lập hóa đơn. Mỗi hóa đơn chỉ dành cho một khách hàng nhưng có thể bao gồm nhiều mặt hàng. Mỗi dòng trên hóa đơn cho biết một mặt hàng được mua với số lượng là bao nhiêu. Dòng cuối cùng trong hóa đơn cho biết tổng số tiền phải trả.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập thông tin danh sách khách hàng và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập thông tin danh sách mặt hàng và lưu vào ArrayList. Hiển thị danh sách ra bảng.
3. Tạo hóa đơn cho một khách hàng bằng cách nhập mã. Thời gian tạo hóa đơn lấy từ thời gian hệ thống. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một mặt hàng với số lượng (kiểu Integer). Nhập từ giao diện các thông tin mua hàng và bổ sung các cặp ghép vào hóa đơn. Hiển thị thông tin chi tiết của hóa đơn ra giao diện
4. Sắp xếp danh sách Hóa đơn theo tên mặt hàng

BÀI 2: QUẢN LÝ NHÂN SU

Thông tin về **Nhân viên** gồm các thuộc tính (*Mã nhân viên, Họ tên, Địa chỉ, Số điện thoại, Bậc lương*) – trong đó Bậc lương là một số nguyên từ 1 đến 9. Mã nhân viên là một số nguyên có 4 chữ số. SĐT là một dãy số gồm 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Phòng ban** (*mã phòng, tên phòng, mô tả, Hệ số công việc*), mã phòng là một số nguyên có 3 chữ số. Hệ số công việc là một số thực từ trong khoảng từ 1 đến 20, cho biết mức đánh giá chung về các công việc thuộc phòng đó đảm nhiệm.

Bổ sung **Bảng Danh sách chấm công** (trong một tháng), trong đó với mỗi nhân viên cho biết phòng ban tương ứng và số ngày làm việc trong tháng (số ngày phải lớn hơn 0 và nhỏ hơn 32).

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách nhân viên lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách phòng ban và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo Danh sách chấm công cho một khách hàng bằng cách nhập mã. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một nhân viên với số ngày làm việc trong tháng (kiểu Integer). Nhập từ giao diện các thông tin chấm công và bổ sung các cặp ghép vào danh sách chấm công. Hiển thị thông tin chi tiết của danh sách hóa đơn ra giao diện.
4. Sắp xếp Bảng danh sách phân công đã lưu Theo tên nhân viên

BÀI 3: QUẢN LÝ TÍNH CÔNG THEO SẢN PHẨM

Thông tin về **Công nhân** gồm các thuộc tính (**mã CN**, **Họ tên**, **Địa chỉ**, **SĐT**, **Ca sản xuất**) – trong đó ca sản xuất có thể là: ca sáng, ca chiều hoặc ca đêm, mã CN là một số nguyên có 5 chữ số. SĐT là một dãy số gồm 9 đến 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Sản phẩm** (**mã SP**, **tên SP**, **nhóm SP**, **đơn giá**), mã SP là một số nguyên có 5 chữ số, ; đơn giá là một số thực không âm.

Bổ sung **Bảng Tính Công** trong đó một công nhân sẽ thực hiện sản xuất nhiều sản phẩm cùng loại hoặc nhiều loại sản phẩm khác nhau.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách công nhân và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách sản phẩm và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo Bảng tính công cho mỗi công nhân. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một công nhân với sản phẩm. Nhập từ giao diện các thông tin về số sản phẩm đã làm trong tháng của một cặp ghép (công nhân, sản phẩm) và bổ sung cặp ghép vào Bảng tính công. Hiển thị thông tin đã nhập ra giao diện
4. Sắp xếp danh sách Bảng tính công theo Họ tên Công nhân

BÀI 4: QUẢN LÝ DỊCH VỤ VIỄN THÔNG

Thông tin về **Khách hàng** gồm các thuộc tính (**mã KH**, **Họ tên**, **Địa chỉ**, **SĐT**, **Loại KH**) – với Loại khách hàng có thể là: cá nhân, đại diện đơn vị hành chính, đại diện đơn vị kinh doanh, mã KH là một số nguyên có 5 chữ số. SĐT là một dãy số có 9 hoặc 10 chữ số. Bắt đầu bằng chữ số 0.

Thông tin về **Dịch vụ** (**mã DV**, **tên DV**, **giá cước**, **đơn vị tính**) mã DV là một số nguyên có 3 chữ số; đơn vị tính là một chuỗi ký tự cho biết đơn vị đo của loại dịch vụ đó (ví dụ theo MB, theo ngày, theo tháng, theo số lượt ...)

Bổ sung **Hóa đơn** trong đó một khách hàng có thể sử dụng một hoặc nhiều dịch vụ.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách khách hàng và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách dịch vụ lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo hóa đơn cho một khách hàng bằng cách nhập mã. Thời gian tạo hóa đơn lấy từ thời gian hệ thống. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một khách hàng với dịch vụ. Nhập từ giao diện các thông tin sử dụng dịch vụ và bổ sung các cặp ghép vào hóa đơn. Hiển thị thông tin chi tiết của hóa đơn ra giao diện
4. Sắp xếp danh sách hóa đơn đã lưu Theo Họ tên khách hàng

BÀI 5: QUẢN LÝ ĐIỂM SINH VIÊN

Thông tin về **Sinh viên** gồm (**mã SV, Họ tên, Địa chỉ, SĐT, Lớp**). Trong đó mã sinh viên là một số nguyên có 5 chữ số. Lớp được đặt tên theo quy tắc tại HV CN BCVT. SĐT có 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Môn học** (**mã môn, tên môn, số tín chỉ, loại môn**). Trong đó, mã môn là một số nguyên có 3 chữ số. Số tín chỉ là một số nguyên dương, không quá 6. Loại môn học có thể là Đại cương, Cơ sở ngành, Chuyên ngành.

Bổ sung **Bảng điểm** trong đó một sinh viên có một điểm cho mỗi môn học ($0 \leq \text{điểm} \leq 10$).

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách sinh viên và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách môn học và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo bảng điểm cho mỗi sinh viên. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một sinh viên với môn học. Nhập từ giao diện điểm tương ứng và bổ sung các cặp ghép vào bảng điểm. Hiển thị thông tin chi tiết của bảng điểm giao diện
4. Sắp xếp danh sách Bảng điểm theo Tên môn học.

BÀI 6: QUẢN LÝ TÍNH CÔNG BÁN HÀNG CHO SIÊU THỊ

Thông tin về **Mặt hàng** gồm các thuộc tính (**mã hàng, Tên hàng, Nhóm hàng, Giá mua, Giá bán**) – trong đó Nhóm hàng có thể là: Hàng thời trang, Hàng tiêu dùng, Hàng điện máy, Hàng gia dụng. Mã hàng là một số nguyên có 4 chữ số. SĐT có 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0. Giá bán luôn lớn hơn giá mua.

Thông tin về **Nhân viên bán hàng** (**mã NV, Họ tên, địa chỉ, Số ĐT**), mã NV là một số nguyên có 4 chữ số.

Bổ sung **Bảng Danh sách bán hàng** trong đó với mỗi nhân viên, nhập danh sách các mặt hàng và số lượng mà nhân viên đó đã bán ra trong ngày.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách nhân viên và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách môn học và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo danh sách bán hàng cho mỗi nhân viên. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một nhân viên với mặt hàng. Nhập từ giao diện các thông tin bán hàng và bổ sung các cặp ghép vào Bảng danh sách mua hàng. Hiển thị thông tin chi tiết ra giao diện.
4. Sắp xếp danh sách Bảng danh sách bán hàng theo tên nhân viên

BÀI 7: QUẢN LÝ CÔNG NHÂN

Thông tin về **Công nhân** gồm các thuộc tính (**mã công nhân, Họ tên, Địa chỉ, Số điện thoại, Bậc thợ**) – trong đó Bậc thợ là một số nguyên từ 1 đến 7. Mã công nhân là một số nguyên có 4 chữ số. SĐT có 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Xưởng sản xuất** (**mã xưởng, tên xưởng, mô tả, Hệ số công việc**), mã xưởng là một số nguyên có 3 chữ số, .. Hệ số công việc là một số thực từ trong khoảng từ 1 đến 20, cho biết mức đánh giá chung về các công việc trong xưởng đó đảm nhiệm.

Bổ sung **Bảng Danh sách chấm công** (trong một tháng), trong đó với mỗi công nhân cho biết đang làm tại xưởng nào tương ứng và số ngày làm việc tương ứng. Một công nhân có thể thay đổi xưởng sản xuất nhiều lần trong tháng nhưng tổng số ngày làm việc không vượt quá 31.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách công nhân và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách xưởng sản xuất lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo bảng chấm công cho từng công nhân. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một công nhân với một xưởng sản xuất. Nhập từ giao diện các thông tin chấm công, trong đó cho biết mỗi công nhân làm trong xưởng sản xuất tương ứng bao nhiêu ngày. Hiển thị thông tin chi tiết ra giao diện.
4. Sắp xếp danh sách Bảng danh sách phân công theo tên công nhân

BÀI 8: QUẢN LÝ BÁN ĐIỆN THOẠI DI ĐỘNG

Thông tin về **Khách hàng** gồm các thuộc tính (**mã KH, Họ tên, Địa chỉ, SĐT, Nhóm KH**) – trong đó nhóm khách hàng có thể gồm: mua lẻ, mua buôn, mua qua mạng, mã KH là một số nguyên có 5 chữ số. SĐT có 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Điện thoại** (**mã sp, hãng sản xuất, phiên bản, đơn giá**), mã sản phẩm là một số nguyên có 5 chữ số, đơn giá là một số thực.

Bổ sung **Hóa đơn** trong đó một khách hàng có thể chọn mua một hoặc nhiều điện thoại với số lượng khác nhau.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách khách hàng và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách điện thoại và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo hóa đơn cho một khách hàng. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một khách hàng với một điện thoại. Nhập từ giao diện các thông tin mua hàng và bổ sung các cặp ghép vào hóa đơn. Hiển thị thông tin chi tiết của hóa đơn ra giao diện.
4. Sắp xếp danh sách mua hàng theo Họ tên khách hàng

BÀI 9: QUẢN LÝ TÍNH LƯƠNG THEO NGÀY CÔNG

Thông tin về **Nhân viên** gồm các thuộc tính (**mã nhân viên, Họ tên, Địa chỉ, Số điện thoại, Bậc lương**) – trong đó Bậc lương là một số nguyên từ 1 đến 9. Mã nhân viên là một số nguyên có 4 chữ số. SĐT có 9 hoặc 10 chữ số, bắt đầu bằng chữ số 0.

Thông tin về **Phòng ban** (**mã phòng, tên phòng, mô tả, Hệ số công việc**), mã phòng là một số nguyên có 3 chữ số, . Hệ số công việc là một số thực từ trong khoảng từ 1 đến 20, cho biết mức đánh giá chung về các công việc thuộc phòng đó đảm nhiệm.

Bổ sung **Bảng Danh sách chấm công** (trong một tháng), trong đó với mỗi nhân viên cho biết phòng ban tương ứng và số ngày làm việc trong tháng.

Tự xác định các lớp cho phù hợp và Viết chương trình trên Java thực hiện các chức năng sau:

1. Nhập danh sách nhân viên và lưu vào ArrayList. Hiển thị danh sách ra bảng.
2. Nhập danh sách phòng ban và lưu vào ArrayList. Hiển thị thông tin ra bảng.
3. Tạo bảng danh sách chấm công cho một nhân viên. Sử dụng lớp Pair viết theo kiểu Generic để ghép cặp một nhân viên với phòng ban. Nhập từ giao diện các thông tin của bảng chấm công và bổ sung các cặp ghép vào bảng. Hiển thị thông tin chi tiết ra giao diện.
4. Sắp xếp danh sách Bảng danh sách phân công theo tên nhân viên