LẬP TRÌNH JAVA CƠ BẢN

Chương 1

GIỚI THIỆU LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lê Tân

Bộ môn: Lập trình máy tính

Tài liệu tham khảo

- Giáo trình "Lập trình Java cơ bản"
- **Trần Tiến Dũng -** *Giáo trình lý thuyết và bài tập Java* **- NXB Giáo dục, 1999**
- **Nell Dale, Chip Weems -** *Java and Software Design* **- Mark Headington, *2001***

Mục đích và yêu cầu

- Làm quen với các cú pháp và phong cách lập trình hướng đối tượng với ngôn ngữ Java.
- Nghiên cứu các mô hình lập trình Java trong việc phát triển các phần mềm hướng đối tượng
- Tạo và biên dịch các chương trình Java như các chương trình ứng dụng
- Tìm và sử dụng các tài liệu chính thức của Java

Nội dung môn học

1. Giới thiệu Lập trình hướng đối tượng
2. Ngôn ngữ lập trình Java
3. Điều khiển sự kiện xuất
4. Các kiểu số và biểu thức
5. Điều khiển sự kiện nhập
6. Điều kiện, biểu thức logic và cấu trúc chọn
7. Lớp và phương thức
8. Thừa kế, đa hình và phạm vi
9. Nhập xuất file, lặp và ngoại lệ
10. Mảng
11. Danh sách và đệ quy

Nội dung chương 1

- Tổng quan về lập trình hướng đối tượng (OOP)
- Các tính chất cơ bản của lập trình hướng đối tượng
- Các ngôn ngữ lập trình hướng đối tượng

1.1 Tổng quan về lập trình hướng đối tượng

- Lập trình tuyến tính:
  - Lập trình bằng assembly
  - Các ngôn ngữ cấp cao lần đầu tiên được sử dụng
  - Các chương trình tương đối ngắn (<100 dòng)
  - Các ngôn ngữ lập trình không còn phù hợp với công việc lập trình đòi hỏi cao hơn

1.1 Tổng quan về lập trình hướng đối tượng

- Lập trình cấu trúc:
  - Vào cuối các năm 1960, đầu 1970, ngôn ngữ lập trình có cấu trúc ra đời
  - Chương trình có cấu trúc được tổ chức theo các công việc mà chúng thực hiện
  - Chương trình chia nhỏ thành các chương trình con riêng rẽ (còn gọi là hàm hay thủ tục) thực hiện các công việc rời rạc trong quá trình lớn hơn, phức tạp hơn.

1.1 Tổng quan về lập trình hướng đối tượng

- Lập trình cấu trúc:
  - Thông tin được chuyển giao giữa các hàm thông qua các tham số
  - Các hàm có thể có các biến cục bộ
  - Các hàm có thể được xem là các chương trình con được đặt chung với nhau để xây dựng nên một ứng dụng
  - Đã có sự trừu tượng hóa theo chức năng (Functional abstraction)

1.1 Tổng quan về lập trình hướng đối tượng

- Nhược điểm của Lập trình cấu trúc:
  - Khi độ phức tạp của một chương trình tăng lên, sự phụ thuộc của nó vào các kiểu dữ liệu cơ bản cũng tăng theo
  - Khi có sự thay đổi trong các dữ liệu này, cần thực hiện các thay đổi mọi nơi có thao tác tác động trên chúng
  - Khi có nhiều lập trình viên làm việc theo nhóm, sai sót trong việc trao đổi thông tin giữa các thành viên có thể dẫn tới hậu quả là mất rất nhiều thời gian để sửa chữa chương trình

1.1 Tổng quan về lập trình hướng đối tượng

- Trừu tượng hóa dữ liệu :
  - Các cấu trúc dữ liệu và các phần tử có thể được sử dụng mà không cần bận tâm đến các chi tiết cụ thể
  - Trừu tượng hóa theo dữ liệu đã tồn tại sẵn trong mọi ngôn ngữ lập trình
  - Gần đây, đã phát triển các ngôn ngữ cho phép chúng ta định nghĩa các kiểu dữ liệu trừu tượng riêng

1.1 Tổng quan về lập trình hướng đối tượng

- Lợi ích của trừu tượng hóa dữ liệu :
  - Tập trung vào vấn đề đang giải quyết
  - Xác định những thuộc tính và hành động thiết yếu
  - Loại trừ những chi tiết không cần thiết

1.1 Tổng quan về lập trình hướng đối tượng

- Lập trình hướng đối tượng (OOP):
  - Được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu
  - Một chương trình hướng đối tượng được thiết kế xoay quanh dữ liệu mà chúng ta có thể làm việc trên đó, hơn là theo bản thân chức năng của chương trình
  - Lập trình hướng đối tượng liên kết cấu trúc dữ liệu với các thao tác trên dữ liệu đó

1.1 Tổng quan về lập trình hướng đối tượng

- Ưu điểm của Lập trình hướng đối tượng (OOP):
  - Tạo ra sự gần gũi giữa bài toán thực tế và việc cài đặt chương trình
  - Đẩy mạnh việc chia sẻ trong phạm vi một ứng dụng
  - Đẩy mạnh sự sử dụng lại của các đối tượng khi cài đặt những ứng dụng mới
  - Về lâu dài, giảm đáng kể chi phí khi phát triển các ứng dụng mới
  - Giảm lỗi và sự phức tạp trong bảo trì
  - Sửa đổi nhanh hơn

1.2 Các tính chất cơ bản của OOP

- Một lớp (Class) là một bản mẫu mô tả các thông tin cấu trúc dữ liệu, lẫn các thao tác hợp lệ của các phần tử dữ liệu
- Một phần tử dữ liệu được khai báo là phần tử của một lớp thì nó được gọi là một đối tượng (Object)
- Các hàm được định nghĩa hợp lệ trong một lớp được gọi là các phương thức (Method)

1.2 Các tính chất cơ bản của OOP

- Lớp định nghĩa một thực thể, còn đối tượng là một thực thể thực tế
- Lớp là một mô hình khái niệm định rõ các thuộc tính và các hành động của một đối tượng, còn đối tượng là một mô hình thực tiễn.
- Lớp là khuôn mẫu của đối tượng
- Tất cả các đối tượng trong cùng một lớp có tập các thuộc tính và hành động như nhau

1.2 Các tính chất cơ bản của OOP

- Đối tượng:
  - Thuộc tính
    - Tính chất mô tả một đối tượng
  - Hành động
    - Dịch vụ mà đối tượng có thể đáp ứng
  - Phương thức
    - Cách hành động được đáp ứng khi được yêu cầu
  - Thông điệp
    - Yêu cầu một hành động
  - Sự kiện
    - Sự kích thích từ đối tượng này gửi sang đối tượng khác

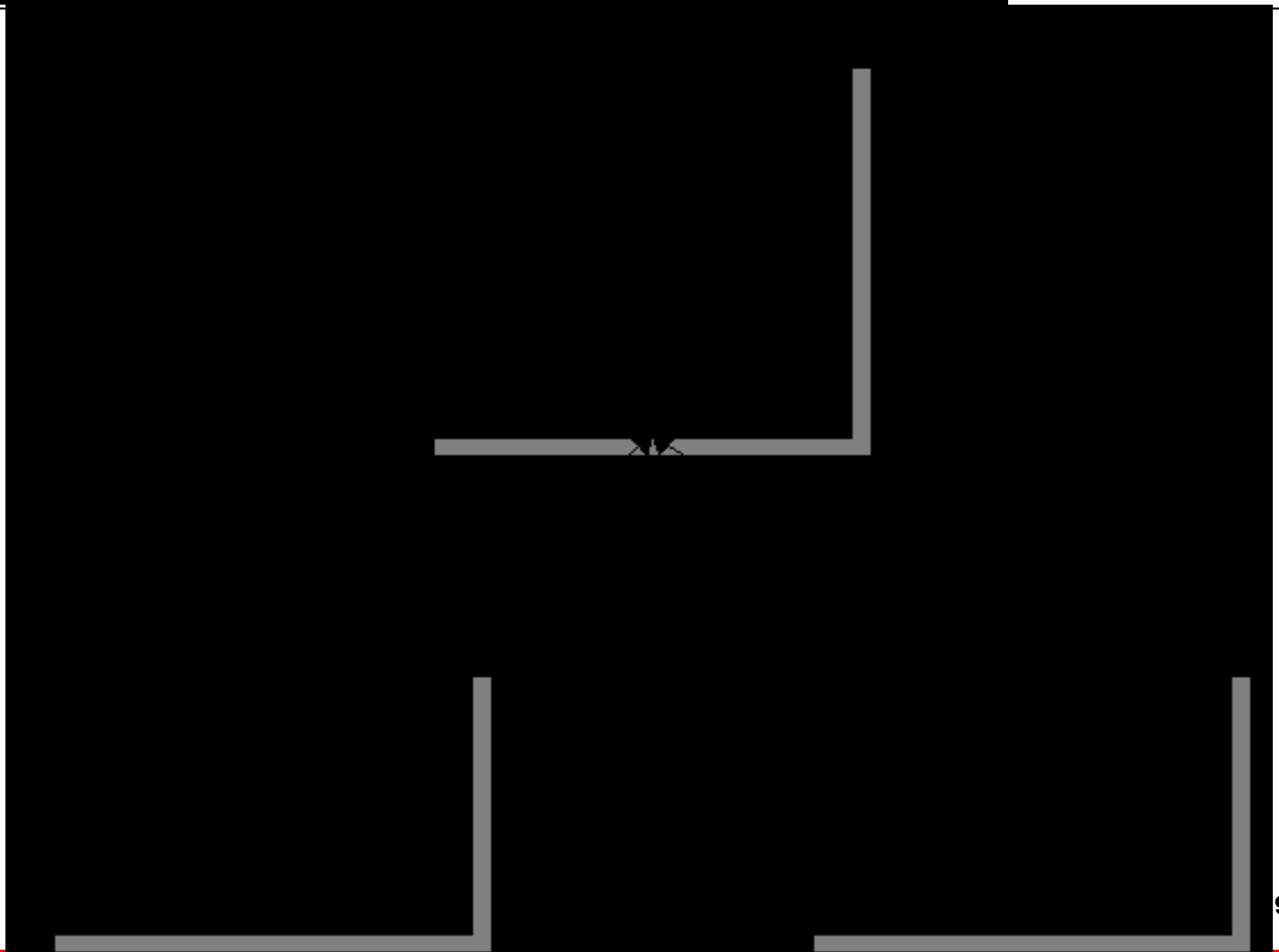
1.2 Các tính chất cơ bản của OOP

- Một thực thể (Instance) là một vật thể bên trong bộ nhớ, đó chính là một đối tượng (được cấp phát vùng nhớ)
- Mỗi đối tượng có riêng một bản sao các phần tử dữ liệu của lớp, còn gọi là các biến thực thể (Instance variable)
- Các phương thức định nghĩa trong một lớp có thể được gọi bởi các đối tượng của lớp đó: gửi một thông điệp (Message) cho đối tượng

1.2 Các tính chất cơ bản của OOP

- Đóng gói (Encapsulation):
  - Cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất
  - Tránh được các tác động bất ngờ từ bên ngoài
- Thừa kế (Inheritance):
  - Xây dựng các lớp mới từ các lớp cũ
  - Lớp mới (lớp dẫn xuất - derived class) có thể thừa kế dữ liệu và phương thức của lớp cơ sở (base class)

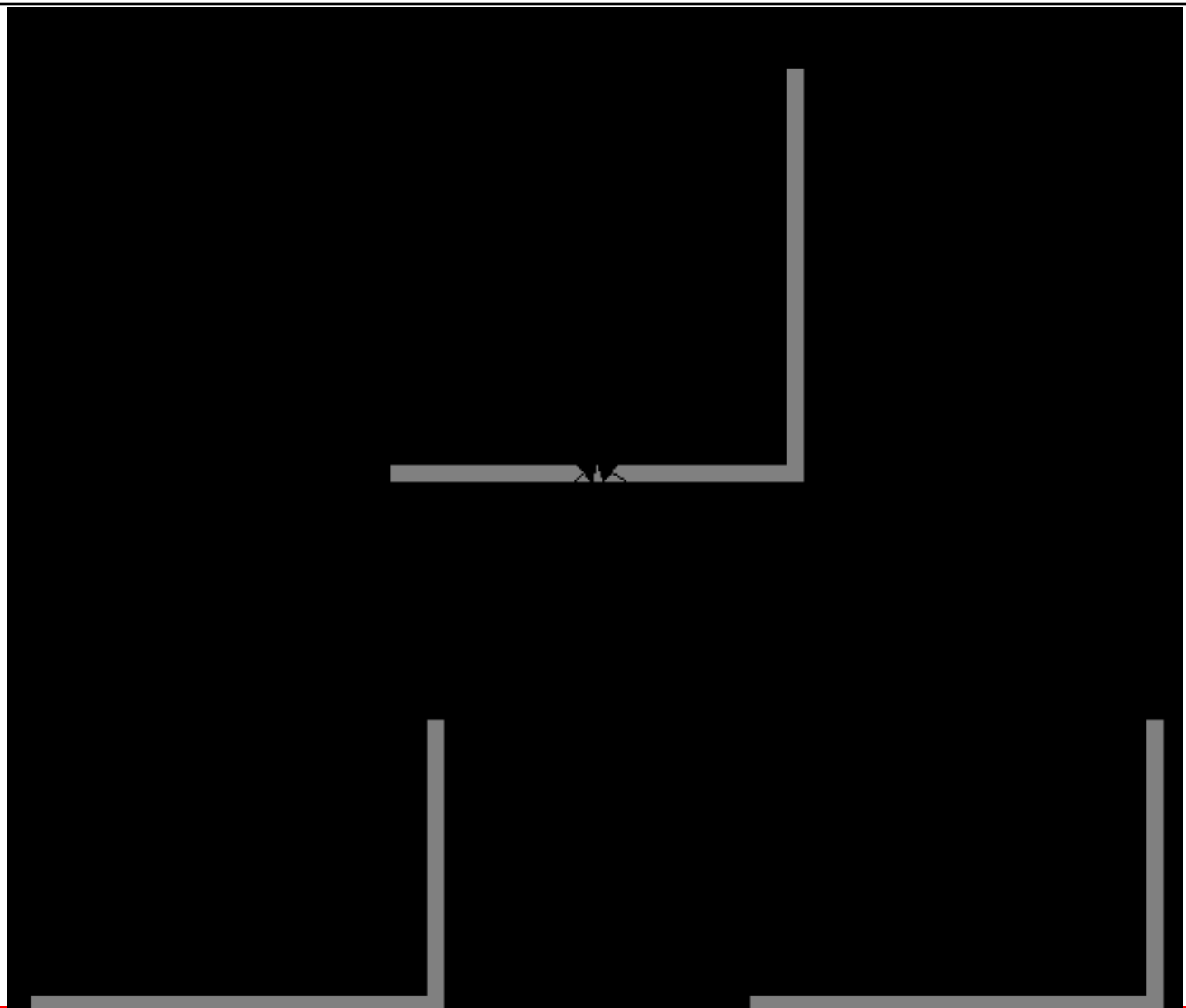
1.2 Các tính chất cơ bản của OOP



1.2 Các tính chất cơ bản của OOP

- Đa hình (Polymorphism):
  - Khả năng của một thông điệp có thể thay đổi cách thực hiện theo lớp cụ thể của đối tượng nhận thông điệp
  - Khi một lớp dẫn xuất được tạo ra, nó có thể thay đổi cách thực hiện các phương thức mà nó thừa kế từ lớp cơ sở
  - Nếu một thông điệp có cùng tên với phương thức này được gởi tới một đối tượng của lớp dẫn xuất sẽ gọi phương thức đã định nghĩa cho lớp dẫn xuất

1.2 Các tính chất cơ bản của OOP



1.3 Các ngôn ngữ OOP

- Ngôn ngữ OOP đầu tiên là Smalltalk, do trung tâm nghiên cứu Palo Alto (PARC) của hãng XEROR tập trung 10 năm nghiên cứu để hoàn thiện từ tư tưởng của ngôn ngữ SIMULA67
- Sau đó các ngôn ngữ OOP lần lượt ra đời như Eiffel, Clos, Loops, Flavors, Object Pascal, Object C, C++, Delphi, Java ...
- Trên cơ sở ngôn ngữ OOP, XEROR đã đề ra tư tưởng giao diện biểu tượng trên màn hình

1.3 Các ngôn ngữ OOP

- Java: là một ngôn ngữ lập trình (*programming language)* hướng đối tượng
- Java hiện đang là một ngôn ngữ rất phổ biến
- Java là một ngôn ngữ mạnh và có tầm bao quát rộng
  - nhưng nó ***không đơn giản***!
- Được so sánh với C++, Java rất "tao nhã" (elegant)

1.3 Các ngôn ngữ OOP Các đặc điểm của Java

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable
- Performance
- Multithreaded
- Dynamic

- đơn giản
- hướng đối tượng
- phân tán
- thông dịch
- mạnh mẽ
- bảo mật
- kiến trúc trung tính
- khả chuyển
- hiệu quả cao
- đa tuyến
- linh động

Câu hỏi và bài tập

1. Thế nào là lập trình tuyến tính? Cho ví dụ.
2. Sự khác nhau giữa lập trình tuyến tính và lập trình có cấu trúc?
3. Lập trình hướng đối tượng? Nêu một vài ngôn ngữ lập trình hướng đối tượng.
4. Tính đóng gói thể hiện trong lập trình hướng đối tượng như thế nào?
5. Thế nào là trừu tượng hóa quá trình và trừu tượng hóa dữ liệu?
6. Tính kế thừa là gì?
7. Tính đa hình thể hiện trong OOP như thế nào?

Chương 2

NGÔN NGỮ LẬP TRÌNH JAVA

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 2

- Tổng quan về Java
- Dịch và thực thi chương trình Java
- Công cụ lập trình và chương trình dịch
- Cú pháp và ngữ nghĩa
- Các kiểu dữ liệu nguyên thủy
- Lớp và đối tượng
- Khai báo và nhập xuất dữ liệu

2.1 Tổng quan về Java

- Java là ngôn ngữ lập trình hướng đối tượng (như C++) do Sun Microsystem đưa ra vào đầu thập niên 90 của thế kỷ 20.
- Chương trình viết bằng Java có thể chạy trên bất kỳ hệ thống nào có cài máy ảo Java (JVM - Java Virtual Machine).

2.1 Tổng quan về Java

- ***Máy ảo Java (JVM - Java Virtual Machine):***
  - **Một chương trình viết bằng Java sẽ được biên dịch ra mã của máy ảo Java (mã java bytecode).**
  - **Máy ảo Java chuyển mã java bytecode thành mã máy tương ứng.**
  - **Sun Microsystem chịu trách nhiệm phát triển các máy ảo Java chạy trên các hệ điều hành trên các kiến trúc CPU khác nhau.**

2.1 Tổng quan về Java

- *Thông dịch:* Chương trình nguồn *.java được biên dịch thành tập tin *.class. Trình thông dịch thông dịch thành mã máy.
- *Độc lập nền:* Một chương trình viết bằng ngôn ngữ Java có thể chạy trên nhiều máy tính có hệ điều hành khác nhau
- *Hướng đối tượng:* Hướng đối tượng trong Java tương tự C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn.

2.1 Tổng quan về Java

- *Đa nhiệm - đa luồng:* Java cho phép nhiều tiến trình có thể chạy song song và tương tác với nhau.
- *Khả chuyển:* Chương trình viết bằng Java có thể chạy được trên bất kỳ máy tính, hệ điều hành nào có máy ảo Java.
- *Hỗ trợ mạnh cho việc phát triển ứng dụng:* Công nghệ Java phát triển mạnh nhờ Sun Microsystem cung cấp nhiều công cụ, thư viện lập trình phong phú

2.1 Tổng quan về Java

Các ứng dụng Java

- **Ứng dụng Console: Ứng dụng nhập xuất ở chế độ văn bản tương tự màn hình Console của MS-DOS.**
- **Ứng dụng Applet: Ứng dụng có thể nhúng và chạy trong trang web của một trình duyệt web.**
- **Ứng dụng Desktop dùng giao diện đồ họa: Phát triển các ứng dụng đồ họa được giải quyết bằng thư viện AWT và JFC.**
- **Ứng dụng Web: Java hỗ trợ mạnh mẽ đối với việc phát triển các ứng dụng Web thông qua công nghệ J2EE (Java 2 Enterprise Edition).**
- **Ứng dụng nhúng: Java Sun đưa ra công nghệ J2ME hỗ trợ phát triển các phần mềm nhúng.**

2.2 ịch và thực thi chương trình Java

- ***Viết mã nguồn:*** Dùng một chương trình soạn thảo để viết mã nguồn, lưu lại với file tên có đuôi ".java". Tên của file phải đặt giống tên của lớp chính trong chương trình.
- ***Biên dịch ra mã máy ảo:*** Dùng trình biên dịch javac để biên dịch mã nguồn ".java" thành mã của máy ảo (java bytecode) có đuôi ".class"
- ***Thông dịch và thực thi:*** Việc thông dịch và thực thi dùng lệnh "java".

2.2 ịch và thực thi chương trình Java

- Ví dụ minh họa: Tạo chương trình nguồn

```
/*Chương trình xuất dòng HelloWorld ra Console*/
package ch01;
import java.util.*;
class HelloWorldApp{
   public static void main(String[] args){
     //Xuat dong chu “HelloWorld”
     System.out.println(“HelloWorld”);
   }
}
```

- Lưu lại với tên HelloWorldApp.java trong thư mục ch01

2.2 ịch và thực thi chương trình Java

- **Dòng đầu tiên khai báo gói chứa chương trình.**
- **Dòng tiếp theo khai báo nạp các lớp sử dụng.**
- **Khai báo lớp *HelloWordApp* phạm vi toàn cục**
- **Phương thức main() là điểm bắt đầu thực thi một ứng dụng.**
- **Lời chú thích: Ngôn ngữ Java hỗ trợ ba kiểu chú thích sau:**
  - **/* text */: Viết chú thích trên nhiều dòng**
  - **// text: Viết chú thích trên một dòng**
  - **/** documentation */: Tự động phát sinh tài liệu.**
- **Dấu "{" và "}": bắt đầu và kết thúc một khối lệnh.**
- **Dấu chấm phẩy ";" để kết thúc một lệnh.**

2.3 Công cụ lập trình và chương trình dịch

- ***J2SE:*** Download J2SE và cài đặt lên máy tính, cần cập nhật đường dẫn PATH hệ thống đến thư mục chứa chương trình dịch của ngôn ngữ java
- ***Viết mã nguồn java:*** Dùng công cụ JCreator LE v 5.0 của hãng XINOX Software.

2.3 Công cụ lập trình và chương trình dịch

- Bước 1-Tạo một dự án rỗng (Empty Project): Chọn menu File \ New \ Project. Chọn Empty project → Next. Nhập tên project, chọn Finish.
- Bước 2-Tạo một lớp mới tên HelloWorldApp và đưa vào Project hiện tại: Chọn File \ New \ Class. Nhập tên lớp là HelloWorldApp và chọn Finish.
- Bước 3-Soạn thảo mã nguồn: Cửa sổ WorkSpace - Cửa sổ soạn thảo mã nguồn. Dịch: Bấm F7. Thực thi: Bấm F5

2.4 Cú pháp và ngữ nghĩa

- Cú pháp: tập các luật xác định chính xác cách kết hợp của các chữ cái, các chữ số, và các ký hiệu.
  - Các luật cú pháp được viết ở dạng đơn giản, xác định ngôn ngữ hình thức, gọi là siêu ngôn ngữ (metalanguage).
- Ngữ nghĩa: tập các luật xác định ý nghĩa các lệnh viết trong một ngôn ngữ lập trình.

2.4 Cú pháp và ngữ nghĩa

- Tên – định danh (Identifier): do người dùng đặt
  - **Là chuỗi các chữ cái, chữ số, dấu gạch dưới (_), và dấu dollar ($).**
  - **Tên phải bắt đầu bởi một chữ cái, dấu gạch dưới (_), hoặc dấu dollar ($)**
  - **Một tên không thể là một từ khóa, không thể là *true*, *false*, hoặc *null*.**
  - **Tên có thể có độ dài bất kỳ.**
- Tên → tham chiếu một lớp, một phương thức, một trường (biến hoặc hằng), hoặc một gói

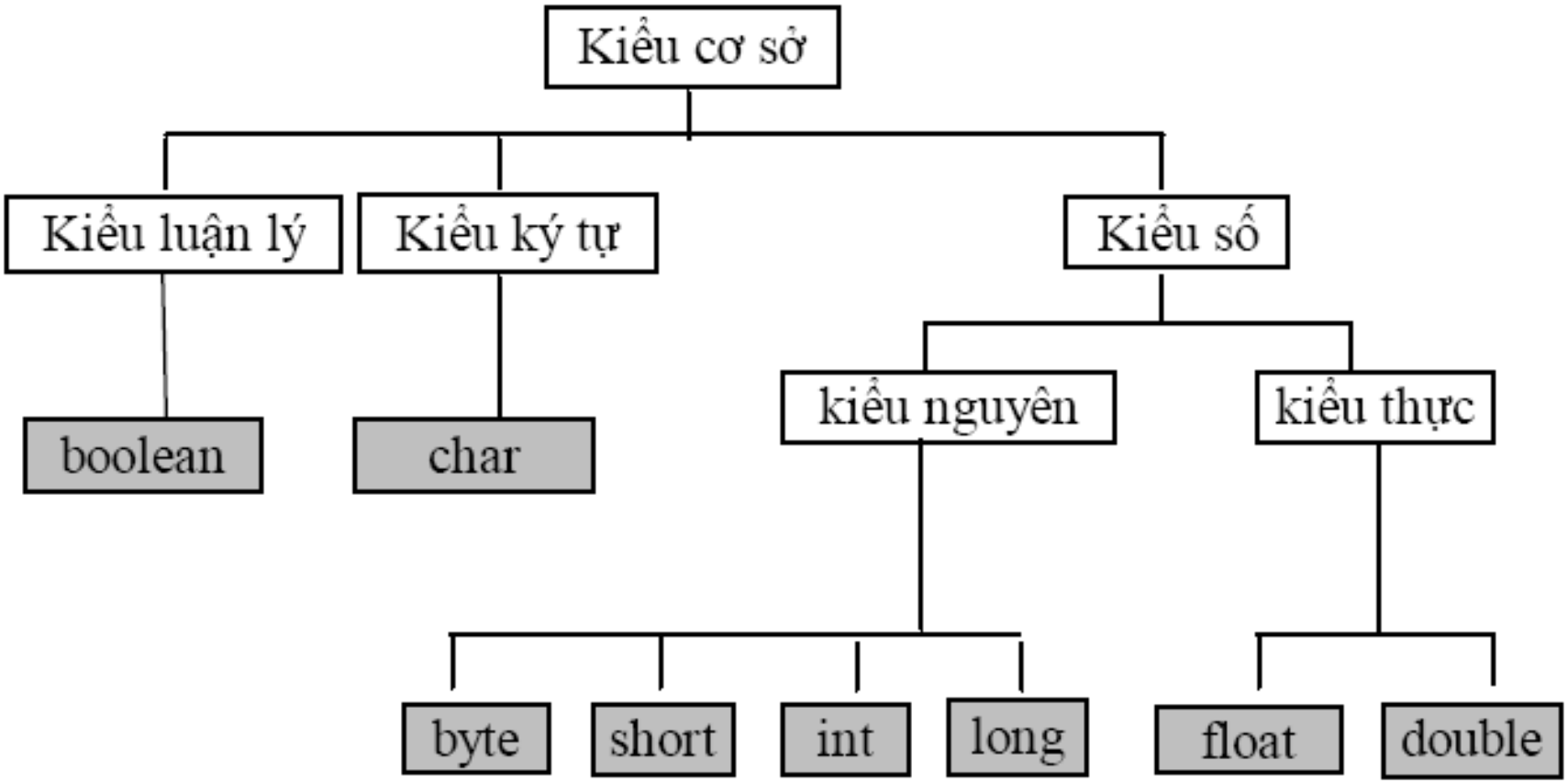
2.4 Cú pháp và ngữ nghĩa

- Các từ khoá của Java

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
false	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
null	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized this	throw	throws	transient
true	try	void	volatile	while

2.5 Các kiểu dữ liệu nguyên thủy

- Là các kiểu dữ liệu có hiệu lực một cách tự động



2.5 Các kiểu dữ liệu nguyên thủy

- Kích thước và miền giá trị

Kiểu	Kích thước (bytes)	Giá trị min	Giá trị max	Giá trị mặc định
byte	1	-256	255	0
short	2	-32768	32767	0
int	4	-2^{31}	$2^{31} - 1$	0
long	8	-2^{63}	$2^{63} - 1$	0L
float	4			0.0f
double	8			0.0d

2.5 Các kiểu dữ liệu nguyên thủy

- Kiểu số nguyên: **byte**, **short**, **int**, **long**. Mặc định là **int**.
- Lưu ý đối với các phép toán trên số nguyên:
  - Nếu hai toán hạng kiểu **long** → kết quả kiểu **long**.
  - Một trong hai toán hạng không phải **long** thì được chuyển thành **long** trước khi thực hiện phép toán.
  - Nếu hai toán hạng đầu không phải kiểu **long** thì phép tính sẽ thực hiện với kiểu **int**.
  - Các toán hạng kiểu **byte**, **short** sẽ được chuyển sang kiểu **int** trước khi thực hiện phép toán.
  - Không thể chuyển biến kiểu **int** và kiểu **boolean**

2.5 Các kiểu dữ liệu nguyên thủy

- ***Kiểu số thực:* float** và **double**
- Không có giá trị nhỏ nhất và lớn nhất. Chúng có thể âm, dương, vô cực âm, vô cực dương.
- Lưu ý đối với các phép toán:
  - Mỗi toán hạng đều có kiểu chấm động thì phép toán chuyển thành phép toán dấu chấm động.
  - Nếu có một toán hạng là **double** thì các toán hạng còn lại → **double** trước khi thực hiện phép toán.
  - Biến kiểu **float** và **double** có thể ép chuyển sang kiểu dữ liệu khác trừ kiểu **boolean**.

2.5 Các kiểu dữ liệu nguyên thủy

- *Kiểu ký tự (char):*
  - Có kích thước là hai bytes
  - Chỉ dùng để biểu diễn các ký tự trong bộ mã Unicode.
  - Như vậy kiểu char trong java có thể biểu diễn tất cả $2^{16} = 65536$ ký tự khác nhau.
  - Giá trị mặc định cho một biến kiểu char là null.

2.5 Các kiểu dữ liệu nguyên thủy

- *Kiểu luận lý (boolean):*
  - Kiểu boolean chỉ nhận một trong hai giá trị: true hoặc false.
  - Trong java kiểu boolean không thể chuyển thành kiểu số nguyên và ngược lại.
  - Giá trị mặc định của kiểu boolean là false.

2.6 Lớp và đối tượng

- Lớp: là một mô tả của một nhóm các đối tượng tương tự nhau về thuộc tính (tập các dữ liệu) và hành vi (tập các phương thức).
- Nói cách khác, lớp là một khuôn mẫu của một đối tượng.
- Ví dụ: lớp hình chữ nhật, là khuôn mẫu của một hình chữ nhật cụ thể.

2.6 Lớp và đối tượng

- Khai báo (định nghĩa) lớp:

  từ_bổ_nghĩa **class** < Tên_lớp>{

  <kiểu dữ liệu> <field_1>;

  …..

  <kiểu dữ liệu> <field_m>;

  constructor

  method_1

  …..

  method_n

  }

2.6 Lớp và đối tượng

- từ_bổ_nghĩa: chỉ phạm vi truy cập của lớp
- **class**: là từ khóa, sử dụng để khai báo lớp
- Tên_lớp: là tên chúng ta đặt cho lớp
- field_1, field_2, …., field_m: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp.
- constructor: phương thức xây dựng, có nhiệm vụ khởi tạo đối tượng.
- method_1, method_2, …, method_n: các phương thức thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

2.6 Lớp và đối tượng

- Đối tượng: một thực thể có liên quan đến ngữ cảnh của một vấn đề.
- Nói cách khác, đối tượng là một trường hợp cụ thể của lớp.
- Ví dụ: Hình chữ nhật ABCD, là một đối tượng của lớp hình chữ nhật.
- Phương thức: xác định một hành vi của một lớp. Ví dụ phương thức *area*, để tính diện tích của hình chữ nhật. Cú pháp tạo đối tượng của lớp:

```
Tên_lớp tên_đối_tượng = new Tên_lớp();
```

2.6 Lớp và đối tượng

- Trong đó:
  - Tên_lớp: là tên của lớp mà đối tượng thuộc về
  - tên_đối_tượng: là tên chúng ta đặt cho đối tượng
  - new: từ khóa, sử dụng để gọi phương thức constructor tạo đối tượng

2.7 Khai báo và nhập xuất dữ liệu

- Khai báo: Một phát biểu để liên kết một tên với một trường, một phương thức, một lớp, hoặc một gói.
- ***Biến:*** Là một vị trí trong bộ nhớ mà chương trình có thể tham chiếu đến qua một tên, lưu trữ một giá trị dữ liệu có thể thay đổi được.
  Khai báo:

  từ_bổ_nghĩa kiểu_dữ_liệu tên1, tên2, . . . ; //hoặc

  từ_bổ_nghĩa kiểu_dữ_liệu tên1 = giá_trị_1, . . . ;

2.7 Khai báo và nhập xuất dữ liệu

- kiểu_dữ_liệu: tên của kiểu hoặc lớp
- từ_bổ_nghĩa: các từ khoá của Java, chỉ phạm vi tác dụng của biến.
- Ví dụ:

  char ch;// Khai báo biến ch kiểu **char**

  public char ch1 = ‘a’;// biến ch1 kiểu **char**, giá trị ban đầu là ‘a’, phạm vi toàn cục

  String firstName; //Khai báo biến firstName thuộc lớp String

- Khai báo biến báo cho máy cấp phát vị trí nhớ đủ lưu trữ một giá trị của kiểu dữ liệu đó.

2.7 Khai báo và nhập xuất dữ liệu

- ***Hằng:*** Là một vị trí trong bộ nhớ mà chương trình có thể tham chiếu đến qua một tên, lưu trữ một giá trị dữ liệu không thay đổi.
- Khai báo hằng:

  từ_bổ_nghĩa **final** kiểu_dữ_liệu tên_hằng = LiteralValue;

- Ví dụ:

  final char ch = ‘a’;// Khai báo hằng ch kiểu **char**, có giá trị là ‘a’

  public final int id = 6;// Khai báo hằng id kiểu **int**, có phạm vi toàn cục

2.7 Khai báo và nhập xuất dữ liệu

- Một chuỗi là một dãy các ký tự đặt giữa hai dấu ngoặc kép. Ví dụ: “Today and tomorrow”, “A”, “”
- Các tác vụ của lớp chuỗi:
- Nối một chuỗi vào một chuỗi khác: toán tử +, ví dụ: String s = “Today” + “and tomorrow”;
- Đổi một giá trị số thành chuỗi: phương thức valueOf của lớp String. Ví dụ: String s = String.valueOf(123);// kết quả s = “123”

2.7 Khai báo và nhập xuất dữ liệu

- Đổi một chuỗi thành số, sử dụng phương thức parseXXX của lớp tương ứng.
- Ví dụ **int** n = Integer.parseInt(“123”);// kết quả n = 123
- So sánh hai chuỗi với nhau, sử dụng phương thức equals của lớp String.
- Ví dụ **boolean** x = “ABC”.equals(“ABC”);// kết quả x = **true**

2.7 Khai báo và nhập xuất dữ liệu

- *Lệnh gán:* được sử dụng để gán biểu thức (expression) cho biến (variable)
- Cú pháp: variable = expression;
- Ví dụ:

```
x = 1;                          // Gán 1 cho x;
bankinh = 1.0;                  // Gán 1.0 cho bankinh;
a = 'A';                        // Gán 'A' cho a;
x = x + 1;                      // Lấy x cộng với 1 rồi gán vào
   x;
c = Math.max(a, b); /* Gán giá trị của lời gọi
   phương thức max(a, b) của lớp Math vào c;*/
```

2.7 Khai báo và nhập xuất dữ liệu

- *Lệnh xuất một chuỗi ra màn hình:* phương thức print, cú pháp như sau:
- System.out.print(StringValue);/* Xuất chuỗi StringValue ra màn hình và không xuống dòng */
- System.out.println(StringValue); /* Xuất chuỗi StringValue ra màn hình và xuống dòng */
- Ví dụ:

```
System.out.print(“The answer is, ”);//Xuất ra chuỗi
  “The answer is, ”
```

2.7 Khai báo và nhập xuất dữ liệu

- *Lệnh nhập một giá trị vào từ bàn phím:*
- Tạo các đối tượng của các lớp từ các lớp nguyên thuỷ hơn
- Sử dụng phương thức nhập dữ liệu.

2.7 Khai báo và nhập xuất dữ liệu

- Ví dụ: /* Khai báo inStream của lớp InputStreamReader*/

```
InputStreamReader inStream;
/* Nhập dữ liệu vào đối tượng inStream */
inStream = new InputStreamReader(System.in);
BufferedReader inData = new
   BufferedReader(inStream);
/* Nhập dữ liệu vào đối tượng inData */
inData = new BuffredReader(new
   InputStreamReader(System.in));
// Lưu trữ một dòng văn bản vào biến oneLine
String oneLine = inData.readLine();
```

2.7 Khai báo và nhập xuất dữ liệu

- Sử dụng đối tượng Scanner: Ví dụ nhập hai số nguyên a và b và xuất ra tổng

```
import java.util.Scanner;
public class InTong{
   public static void main(String[] args){
     Scanner s = new Scanner(System.in);
     int a = s.nextInt();          int b = s.nextInt();
     int c = a + b;
     System.out.print("Tong la:" + c);
   }
}
```

Câu hỏi và bài tập

1. Sự khác nhau giữa mã máy và mã Byte?
2. Các công đoạn máy ảo Java thực hiện trong việc biên dịch và thông dịch là gì?
3. Viết chương trình xuất dòng chữ “Welcome to Java” ra màn hình.
4. Mọi chương trình Java đều chứa ít nhất bao nhiêu phương thức?
5. Hãy viết một khai báo hằng Java để tên *ZED* sẽ cho giá trị ‘z’.
6. Hãy khai báo một biến *char* tên là *letter* và một biến *String* tên *street*.

Câu hỏi và bài tập

7. Có bao nhiêu ký tự có thể được lưu trữ trong một biến kiểu *char*?
8. Có bao nhiêu ký tự trong một xâu (string) rỗng?
9. Một biến thuộc lớp *String* có thể được gán vào một biến biến kiểu *char*, đúng hay sai?
10. Sự khác nhau giữa chuỗi tự nhiên "*computer*" và định danh "*computer*" là gì?

Câu hỏi và bài tập

11. Đầu ra của đoạn mã sau là gì? (Tất cả các biến đều thuộc lớp *String*)

```
street = “Elm St.”;
address = “142B”;
city = “Amaryllis”;
state = “Ioma”;
streetAddress = address + “ “ + street;
System.out.println(streetAddress);
System.out.println(city);
System.out.println(“,” + state);
```

LẬP TRÌNH JAVA CƠ BẢN

Chương 3

ĐIỀU KHIỂN SỰ KIỆN XUẤT

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 3

- Giao diện và lập trình điều khiển sự kiện
- Định dạng xuất
- Quản lý sự kiện
- Đăng ký listener
- Các phương thức event-handler

3.1 Giao diện và lập trình điều khiển sự kiện

- *Giao diện (interface):* Là mặt kết nối, cho phép các hệ thống độc lập gặp gỡ và tác động hoặc giao tiếp với nhau.
- *Lập trình điều khiển sự kiện (event-driven programming):* Tương tác của người sử dụng với một thành phần của giao diện đồ hoạ người dùng GUI là một sự kiện có thể được xử lý bởi chương trình.
- *Frame (khung):* Một kiểu cửa sổ được người sử dụng định nghĩa gọi là một khung.

- Thực hiện các bước:
  - Nạp gói (package – tương tự khái niệm thư mục, hay folder của Windows) chứa lớp *Frame* từ thư viện Java, đó là gói java.awt
  - Khai báo một biến thuộc lớp *Frame*
  - Sử dụng toán tử **new** để tạo một đối tượng thuộc lớp *Frame* và gán cho biến đã khai báo

3.1 Giao diện và lập trình điều khiển sự kiện

- Xác định bộ quản lý bố cục (layout manager) cho đối tượng *Frame* đó
- Thêm phần tử xuất dữ liệu (ví dụ một nhãn – label) vào đối tượng *Frame*.
- Điều chỉnh kích thước của đối tượng *Frame* cho phù hợp với dữ liệu xuất mà nó chứa.
- Hiển thị đối tượng *Frame* trên màn hình

3.1 Giao diện và lập trình điều khiển sự kiện

- **Ví dụ về sử dụng khung xuất**

```
import  java.awt.*;       // Nạp gói java.awt
.....
private static Frame outputDisplay;//đối tượng
   Frame
outputDisplay = new Frame( );  // Khởi tạo đối
   tượng
   // Xác định bộ quản lý bố cục cho khung
   outputDisplay.setLayout( new FlowLayout( ));
   //Thêm nhãn để hiển thị dữ liệu vào khung
   outputDisplay.add( new Label("Total is $" + total));
   outputDisplay.pack( ); // Điều chỉnh kích thước
   outputDisplay.show( ); // Hiển thị đối tượng Frame
```

3.1 Giao diện và lập trình điều khiển sự kiện

- Phương thức tạo *(Constructor)*:
  - *Constructor* là một dạng đặc biệt của phương thức
  - Được gọi để xây dựng đối tượng.
  - Tên của nó cần phải đặt giống với tên của lớp chứa nó.
  - *Constructor* không có tham số được gọi là *default constructor*.
  - Các *Constructor* không có kiểu dữ liệu trả về, kể cả kiểu *void*.
  - Các *Constructor* được gọi sử dụng toán tử *new* khi tạo một đối tượng, đóng vai trò khởi tạo đối tượng.
- Ví dụ về *constructor:*

```
//  Constructors
public Label( )
//  Tạo một nhãn rỗng (không chứa văn bản)
```

3.1 Định dạng xuất

- GUI (Abstract Windowing Toolkit package – gói công cụ cửa sổ trừu tượng): Giao diện đồ họa người sử dụng.
- Một số thành phần của GUI:
  - *Khung (frame):* Một dạng cửa sổ (window) chứa các thành phần đồ họa khác.
  - *Nhãn (label):* Thành phần hiển thị văn bản
  - *Nút (button):* Thành phần tạo ra một sự kiện
  - *Trường văn bản (textfield):* Thành phần trong đó người sử dụng có thể nhập một giá trị.

3.1 Định dạng xuất

- *Bộ quản lý bố cục: Định dạng bảng cho việc xuất dữ liệu*
- *FlowLayout:*
  - Chương trình quản lý bố cục đơn giản nhất
  - Quản lý mặc định cho khung
  - Đặt các thành phần cái nọ sau cái kia.
- *GridLayout:*
  - Rất giống với *FlowLayout*.
  - Chia nhỏ khung thành một khối chữ nhật với số hàng và số cột cố định
  - Chèn các thành phần vào các ô lưới, hết hàng này đến hàng kia.

3.1 Định dạng xuất

- **Định dạng xuất văn bản trong nhãn**
  - Mặc định, văn bản được căn lề trái.
  - Có thể căn lề phải, hoặc căn giữa cho văn bản trong nhãn.
  - Label.LEFT, Label.CENTER, Label.RIGHT.
- Ví dụ: Muốn cột "Named" trong bảng trên được canh giữa:

```
dateWindow.add(new Label("Named",
    Label.CENTER));
```

3.1 Quản lý sự kiện

- Event (sự kiện): Là một hành động, chiếm một vị trí không đồng bộ theo khía cạnh thực hiện chương trình.
- Event handling (xử lý sự kiện): Tiến trình đáp ứng các sự kiện.
- Event listener (bộ nghe sự kiện): Đối tượng, chờ đợi sự xuất hiện của một hoặc nhiều sự kiện.
- Event handler (bộ xử lý sự kiện): Phương thức, là bộ phận của event listener, được gọi khi listener nhận được một sự kiện tương ứng.
- Event source (nguồn sự kiện): Đối tượng sản sinh sự kiện.
- Firing an event: Việc sản sinh sự kiện của một event source.
- Registering the listener (đăng ký bộ nghe sự kiện): Thêm listener vào danh sách các listener được quan tâm của event source.

3.1 Quản lý sự kiện

- Các bước xử lý một sự kiện
  - Ghi nhận của một event listener để nhận biết dạng đặc trưng của sự kiện
  - Việc thi hành event handler được gọi tự động nhằm đáp ứng cho một dạng riêng biệt của sự kiện

3.1 Quản lý sự kiện

- Chế độ uỷ quyền xử lý sự kiện
  - Sử dụng event listener trong việc quản lý sự kiện
  - Việc xử lý một sự kiện được uỷ quyền cho một đối tượng đặc biệt trong chương trình
  - Sử dụng một lớp riêng biệt để định nghĩa event listener là thực tế thường gặp để tách giao diện GUI từ việc thi hành của event listener của nó

3.4 Đăng ký listener

- Việc đăng ký một event listener được thực hiện bằng cách gọi một phương thức liên kết với đối tượng event source.
- Phương thức đăng ký một listener là *add*, theo sau là tên listener.
- Ví dụ: listener của các sự kiện window là *WindowListener* → phương thức đăng ký listener là *addWindowListener*. Được thực hiện như sau:

```
outputDisplay.addWindowListener (myListener);
```

3.4 Đăng ký listener

- *Các lớp thích nghi:* Có thể một chương trình chỉ cần một số ít các phương thức từ một giao diện event listener
- Sẽ có các lớp thích nghi với phần thân rỗng cho mỗi phương thức có trong giao diện tương ứng.
- Ví dụ: *lớp WindowAdapter*.
- Ưu điểm: chỉ định nghĩa các phương thức thật sự cần thiết cho việc quản lý sự kiện.

3.4 Các phương thức event handler

```
//Ví dụ: Xuất ra một ngày theo 2 định dạng khác nhau
import  java.awt.*;     // Nạp lớp trong gói java.awt
import  java.awt.event.*;
public class DateFormats {
  private static Frame outputDisplay;   //Khai báo đối tượng
  public static void main( String[ ] args )  {
     final String  MONTH_NAME = “August”; // Tên tháng
     final String  MONTH_NUMBER = “8”;    // Tháng theo số
     final String  DAY = “17”;            // Ngày trong tháng
     final String  YEAR = “2001”;         // Năm theo 4 ký số
     String  first;         //Ngày theo dạng tháng – ngày – năm
     String  second;    // Ngày theo dạng tháng – năm – ngày
     outputDisplay = new Frame( );
     outputDisplay.setLayout( new GridLayout(5, 2) );
```

3.4 Các phương thức event handler

```
outputDisplay.add( new Label(“Format”));
outputDisplay.add( new Label(“Example”));
// Hiển thị ngày theo dạng thứ nhất
outputDisplay.add( new Label(“Month day, year”));
first = MONTH_NAME + “ ” + DAY + “, ” + YEAR;
outputDisplay.add( new Label(first) );
// Hiển thị ngày theo dạng thứ hai
outputDisplay.add( new Label(“day Month year”));
second = DAY + “ ” + MONTH_NAME + “ ” + YEAR;
outputDisplay.add( new Label(second) );
outputDisplay.pack( ); //Điều chỉnh kích thước khung
outputDisplay.show( );// Hiển thị khung
```

3.1 Các phương thức event handler

```
    // Quản lý sự kiện cửa sổ đóng

    outputDisplay.addWindowListener( new WindowAdapter( )
    // Tạo một đối tượng WindowAdapter
       {
          // Phương thức thay thế phương thức rỗng
          public void windowClosing (WindowEvent event)
          {
             outputDisplay.dispose( );  // Xóa khung
             System.exit( 0 );          // Thoát chương trình
          }
       });
  }
}
```

Câu hỏi và bài tập

1. Từ khoá nào của Java được sử dụng để nạp vào các lớp trong một gói?
2. Viết lệnh tạo một đối tượng Frame tên là *outDisplay*.
3. Thành phần được sử dụng để truyền các giá trị đến một phương thức là gì?
4. Có bao nhiêu cột và bao nhiêu hàng trong một khung được định dạng theo lệnh sau?

```
out.setLayout(new GridLayout(4, 3));
```

Câu hỏi và bài tập

5. Hãy viết một lệnh để thêm nhãn “This is my answer” vào *outDisplay*.
6. Chức năng của phương thức *add* của lớp *Frame* là gì?
7. Mục đích của phương thức *setLayout* là gì? Hãy gọi tên hai chương trình quản lý bố cục và nêu sự khác nhau giữa chúng.
8. Đối tượng mà vai trò của nó là chờ đợi sự xuất hiện của một sự kiện là gì?
9. Đoạn mã của phương thức *windowClosing* được chứa ở đâu?
10. Hãy sửa đổi chương trình *DateFormats* để hiển thị ngày sinh của bạn.

Chương 4

CÁC KIỂU SỐ VÀ BIỂU THỨC

Lê Tân

Bộ môn: Lập trình máy tính

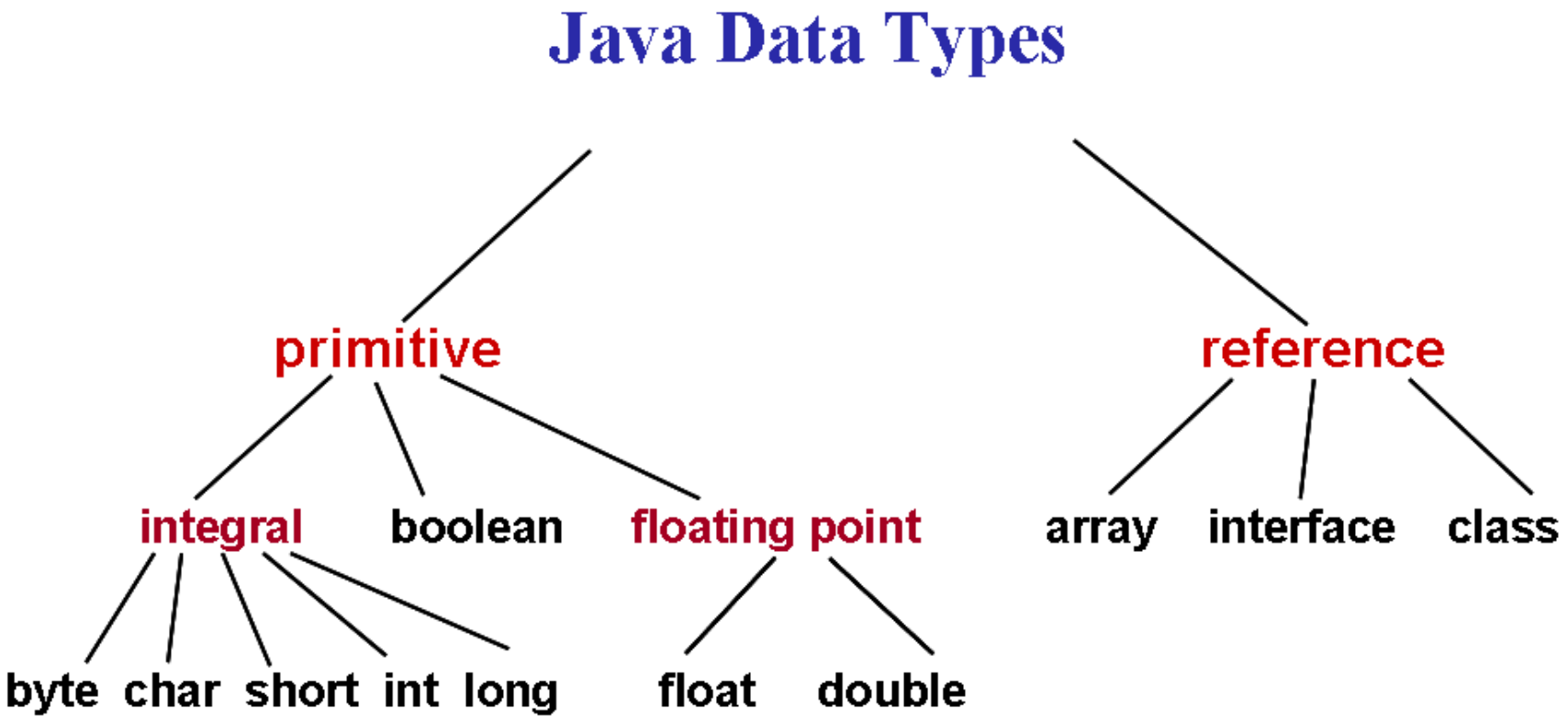
Nội dung chương 4

- Tổng quan về các kiểu dữ liệu Java
- Các kiểu dữ liệu số
- Khai báo các kiểu số
- Các biểu thức số học đơn giản
- Chuyển đổi kiểu (ép kiểu)
- Các phương thức của lớp Math (toán học)
- Các phương thức của lớp String (chuỗi)

4.1 Tổng quan về các kiểu dữ liệu Java

- *Các kiểu dữ liệu Java:* Có hai dạng là dạng nguyên thuỷ và dạng tham chiếu.
  - Dạng nguyên thuỷ: Gồm các kiểu số nguyên, kiểu *Boolean*, và kiểu số thực.
  - Dạng tham chiếu: Chứa số các phần tử khác nhau, thường là quá lớn để có thể chứa vừa trong một vị trí nhớ đơn.
    - Java lưu đối tượng đó vào một hoặc vài vị trí trong phần khác của bộ nhớ
    - Vị trí đã chọn cho biến tham chiếu sẽ lưu địa chỉ mà ở đó đối tượng có thể được tìm thấy.

4.1 Tổng quan về các kiểu dữ liệu Java



4.2 ác kiểu dữ liệu số

- Kiểu số nguyên (integral number)
  - Kiểu số nguyên có thể biểu diễn các số nguyên khi khai báo với từ khoá **byte**, **short**, **int**, hoặc **long**;
  - Cũng có thể biểu diễn các ký tự đơn khi khai báo với từ khoá **char**.
  - Các kiểu số nguyên có độ dài như sau:

byte	có độ dài	8 bits
short	có độ dài	16 bits
int	có độ dài	32 bits
long	có độ dài	64 bits

4.2 ác kiểu dữ liệu số

- Kiểu số thực: biểu diễn các số thực với dấu chấm thập phân, khai báo với từ khoá **float**, hoặc **double**.
- Độ dài như sau:

  float có độ dài 32 bits

  double có độ dài 64 bits
- Kiểu chấm động gồm có phần nguyên và phần thập phân. Ví dụ: 18.4; 500.; .8; -127.358
- Kiểu chấm động có thể có phần mũ, ở dạng khoa học. Phần số sau ký tự E không có dấu chấm thập phân. Ví dụ: 1.84E1, 5E2, 8E-1, -.127358E3

4.3 Khai báo ác kiểu số

- **Khai báo hằng**
  - *Cú pháp: Modifiers final dataType name = value;*
  - *Trong đó: Modifiers* là phần tuỳ chọn, xác định phạm vi tác dụng của hằng; *final* là từ khoá bắt buộc; *dataType* xác định kiểu dữ liệu của hằng; *name* là tên hằng; *value* là giá trị của hằng.
  - Ví dụ:

```
final double    PI = 3.14159;  //Hằng số thực PI
final float     E = 2.72;      // Hằng số thực E
final long      MAX_TEMP = 1000000L; //Hằng nguyên
final int       MIN_TEMP = -273; //Hằng số nguyên
final char      LETTER = 'W'; //Hằng ký tự
final String    NAME = "Elisa"; //Hằng chuỗi ký tự
```

4.3 Khai báo ác kiểu số

- Khai báo biến
  - Cú pháp: Modifiers dataType name1, name2, . . . ;
  - Trong đó: Modifiers là phần tuỳ chọn; dataType xác định kiểu dữ liệu của biến; name1, name2, … là tên biến.
  - Ví dụ:

```
int     studentCount, age;//Các biến studentCount, age
long    sumofSquares; //Khai báo biến sumofSquares kiểu long
String  stuName; //Khai báo biến stuName kiểu String
```

  - Khai báo biến và khởi tạo giá trị trong 1 lệnh:

Modifiers dataType name1= value1, name12 = value 2, . . . ;

Ví dụ:

```
int     i = 1, j = 5;
double  d = 1.4;
float   pi = 3.1416f;
```

4.4 Các biểu thức số học đơn giản

- Biểu thức:
  - Là một dãy các lời gọi phương thức, biến, hằng, các toán tử và dấu ngoặc
  - Có thể được ước lượng để tính một giá trị thuộc một kiểu dữ liệu cho trước.
- Biểu thức số học:
  - Là một dãy các lời gọi phương thức kiểu số, biến và hằng kiểu số, các toán tử số học và dấu ngoặc.
  - Một biểu thức số học có thể được ước lượng để tính một giá trị thuộc kiểu dữ liệu số cho trước, ví dụ giá trị của biểu thức 9.3 * 4.5 là 41.85, và thuộc kiểu số thực.

4.4 Các biểu thức số học đơn giản

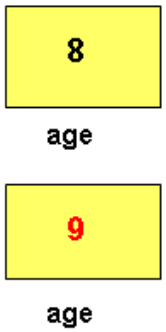
- Một số toán tử Java và độ ưu tiên của chúng theo thứ tự giảm dần:
  - Dấu ngoặc ()
  - +, - (dấu dương, âm)
  - (type) Casting (ép kiểu)
  - *, /, % (nhân, chia thường, chia lấy phần dư)
  - +, - (cộng, trừ)
  - = (phép gán)

4.4 Các biểu thức số học đơn giản

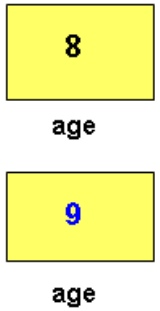
- Các toán tử khác:
  - Toán tử tăng dạng tiền tố: Tăng giá trị của biến trước, sử dụng sau.
  - Toán tử tăng dạng hậu tố: Sử dụng trước, tăng giá trị của biến sau.
  - Toán tử giảm dạng hậu tố: Sử dụng trước, giảm giá trị của biến sau.
  - Toán tử tăng/giảm dạng tiền tố: Tăng/giảm trước, sử dụng sau
  - Toán tử tăng/giảm dạng hậu tố: Sử dụng trước, tăng/giảm sau

4.4 Các biểu thức số học đơn giản

```
int age;

age = 8;

++age;
```

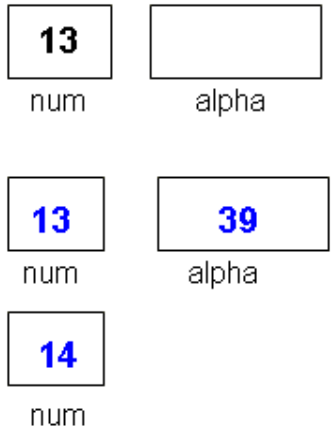



```
int age;

age = 8;

age++;
```




```
int dogs;

dogs = 100;

dogs--;
```


```
int alpha ;
int num ;

num = 13;

alpha = num++  * 3;
```



4.5 Chuyển đổi kiểu (ép kiểu)

- ***Phép gán:***
  - Cú pháp: tên_biến = biểu_thức;
  - Biểu thức bên phải sẽ được tính toán
  - Kết quả được lưu vào vị trí nhớ của biến bên trái
  - Nếu kiểu của biểu thức và biến khác nhau → chuyển kiểu tự động sau khi tính giá trị của biểu thức nhưng trước khi giá trị đó được lưu.

4.5 Chuyển đổi kiểu (ép kiểu)

- Luật chuyển kiểu:
  - Nếu một toán hạng kiểu double, toán hạng khác được chuyển đổi thành kiểu double.
  - Nếu không thì, nếu một toán hạng kiểu float, toán hạng khác được chuyển đổi thành kiểu float.
  - Nếu không thì, nếu một toán hạng kiểu long, toán hạng khác được chuyển đổi thành kiểu long.
  - Nếu không thì cả hai toán hạng được chuyển đổi thành kiểu int.

4.5 Chuyển đổi kiểu (ép kiểu)

- Ép kiểu mở rộng: Là sự chuyển kiểu không làm mất thông tin.
  - Ví dụ: phép gán double someDouble = 3 sẽ thực hiện chuyển kiểu số nguyên thành số thực nhưng không làm mất thông tin.
- Ép kiểu thu hẹp: Là sự chuyển kiểu có mất thông tin
  - Ví dụ:

    int someInt = (int)3.6; //(thu hẹp kiểu)
- Chuyển kiểu tường minh (Type casting): Là sự chuyển kiểu một cách tường minh, do người lập trình xác định.
  - Ví dụ: phép gán int someInt = (int) (5.2 / someDouble) → chuyển giá trị biểu thức từ số thực sang số nguyên.

4.6 Các phương thức của lớp Math (toán học)

- Một lời gọi phương thức sẽ tạm thời chuyển điều khiển đến mã phương thức được gọi để thực hiện.
- Khi mã của phương thức đó đã được thực hiện xong, điều khiển sẽ được trả về cho khối gọi.
- Các phương thức của Java được lưu trữ trong các gói (tương tự khái niệm folder của Windows), hoặc được viết bởi người sử dụng
- Phương thức toán học là các phương thức được sử dụng để tính toán các hàm toán học và được khai báo trong lớp *Math*.

4.6 Các phương thức của lớp Math (toán học)

- Phương thức trả về một giá trị (value-returning): Được gọi như một phần của biểu thức, thực hiện một nhiệm vụ nào đó và trả về một giá trị.
- Phương thức không trả về giá trị (void): Được gọi như một chỉ thị (lệnh) riêng biệt, thực hiện một nhiệm vụ nào đó và không trả giá trị về cho bộ phận gọi nó.
- *Cú pháp gọi phương thức:* tên_phương_thức(danh sách đối số)
  - Danh sách đối số được sử dụng để trao đổi các giá trị với phương thức qua việc truyền thông tin.
  - Danh sách đối số có thể chứa 0, 1 hoặc nhiều hơn các đối số, và được tách biệt nhau bởi các dấu phẩy (,)

4.6 Các phương thức của lớp Math (toán học)

Phương thức	Ý nghĩa
sin(double rad)	Trả về giá trị sin của rad (rad theo radian)
cos(double rad)	Trả về giá trị cos của rad (rad theo radian)
tan(double rad)	Trả về giá trị tan của rad (rad theo radian)
acos(double a)	Trả về giá trị acos của a (tính theo radian)
asin(double a)	Trả về giá trị asin của a (tính theo radian)
atan(double a)	Trả về giá trị atan của a (tính theo radian)
toRadians(double deg)	Đổi từ độ sang radian
toDegrees(double rad)	Đổi từ radian sang độ
exp(double a)	Trả về e^a
log(double a)	Trả về ln(a)
pow(double a, double b)	Trả về a^b
sqrt(double a)	Trả về căn bậc hai của a
double ceil(double x)	Làm tròn x lên giá trị nguyên gần nhất
double floor(double x)	Làm tròn x xuống giá trị nguyên gần nhất
double rint(double x)	Làm tròn x đến giá trị nguyên gần nhất
int round(float x)	Trả về (int)Math.floor(x+0.5)
long round(double x)	Trả về (long)Math.floor(x+0.5)
max(a, b)	Trả về giá trị lớn nhất của hai số a, b
min(a, b)	Trả về giá trị nhỏ nhất của hai số a, b
abs(a)	Trả về giá trị tuyệt đối của a
double random()	Trả về giá trị ngẫu nhiên thuộc [0.0, 1.0)

4.1 Các phương thức của lớp String (chuỗi)

- Phương thức *length:* trả về số các ký tự có trong chuỗi.
  - Ví dụ: Nếu *myName* là một đối tượng chuỗi, ta gọi *myName.length*();
- Phương thức *indexOf:* trả về một số nguyên là vị trí bắt đầu của lần xuất hiện đầu tiên của chuỗi con trong chuỗi đã cho. Nếu không tìm thấy chuỗi con, phương thức trả về giá trị -1.
  - Ví dụ: Để tìm vị trí của lần xuất hiện đầu tiên của ký tự "a" trong *myName*, ta gọi *myName.indexOf*("*a*");
- Phương thức *substring:* trả về chuỗi con của một chuỗi. Tham số đầu tiên chỉ ra vị trí bắt đầu trong chuỗi. Tham số thứ hai lớn hơn 1 so với vị trí kết thúc của chuỗi con.
  - Ví dụ: Để lấy ra các ký tự thứ hai và thứ ba của *myName*, ta gọi *myName.substring*(1, 3);

4.1 Các phương thức của lớp String (chuỗi)

- Ví dụ về các phương thức *length, indexOf, substring*

```
String  stateName = “Mississippi” ;
stateName.length( )
stateName.indexOf(“is”)
stateName.substring( 0, 4 )
stateName.substring( 4, 6 )
stateName.substring( 9, 11 )
stateName.indexOf(“si”)
stateName.indexOf(“as”)
```

Câu hỏi và bài tập

1. Hãy viết một khai báo hằng Java cục bộ để truyền giá trị 3.14159 cho tên PI.
2. Hãy khai báo một biến *count* kiểu *int* và một biến *sum* kiểu *double*.
3. Ta muốn chia 9 cho 5.

   Ta sẽ viết biểu thức như thế nào nếu muốn kết quả là giá trị chấm động?

   Ta sẽ viết biểu thức như thế nào nếu muốn kết quả là phần nguyên của thương?
4. Giá trị của biểu thức Java sau là gì?

   5 % 2

Câu hỏi và bài tập

1. Kết quả của phép tính biểu thức sau là gì?

   (1 + 2 * 2) / 2 + 1

2. Nếu *alpha* và *beta* là các biến *int* với *alpha* = 4 và *beta* = 9, giá trị nào được lưu trữ vào *alpha* sau mỗi lệnh dưới đây? Các lời giải của mỗi phần là độc lập lẫn nhau.

   alpha = 3 * beta;

   alpha = alpha + beta;

   alpha ++;

   alpha = alpha / beta;

   alpha --;

   alpha = alpha + alpha;

   alpha = beta % 6;

3. Hãy tính giá trị của các biểu thức hợp lệ dưới đây và nói rõ giá trị đó là số nguyên hay số chấm động. Nếu biểu thức là chưa hợp lệ, hãy giải thích tại sao.

4. Hãy chuyển mã Java sau sang dạng đại số. (Tất cả các biến đều thuộc kiểu *double*)

   y = -b + sqrt(b * b – 4.0 * a * c);

Câu hỏi và bài tập

1. Hãy viết một lệnh gán để tính tổng của các số từ 1 đến *n* sử dụng công thức Gauss sau:

   sum = n(n + 1) / 2

2. Hãy viết chương trình chuyển đổi độ Celsius sang độ Fahrenheit theo công thức sau:

   Fahrenheit = 9/5 Celsius + 32

   Trong đó độ Celsius là hằng. Chương trình sẽ xuất ra hai giá trị là độ Celsius và độ Fahrenheit tương ứng.

3. Viết chương trình tính đường kính, chu vi và diện tích của một đường tròn bán kính 6.75. Khai báo hằng PI với giá trị 3.14159. Chương trình sẽ đưa ra đường kính, chu vi, và diện tích trên từng dòng riêng biệt, cùng với nhãn của chúng. Mỗi giá trị trên có năm vị trí phần thập phân trong một trường chung có độ dài là mười.

Chương 5

ĐIỀU KHIỂN SỰ KIỆN NHẬP

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 5

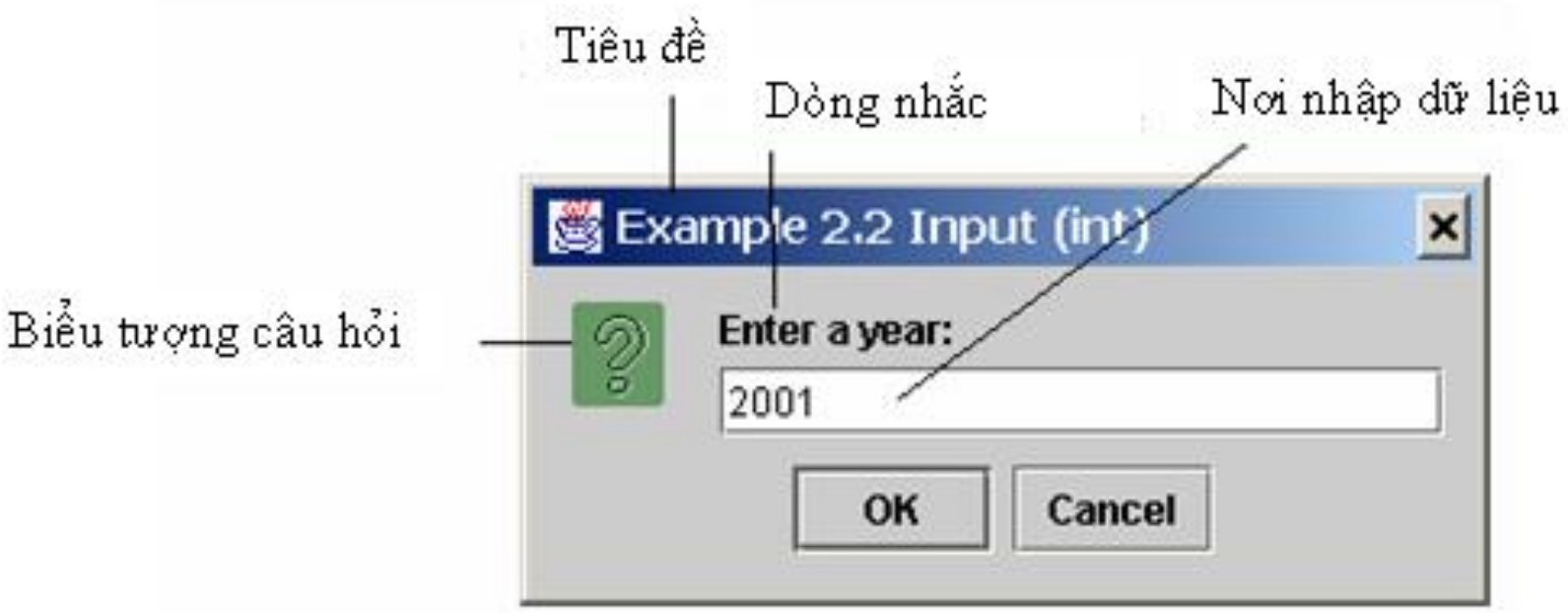
- Nhập dữ liệu từ hộp hội thoại
- Nhập dữ liệu sử dụng Frame
- Tạo một trường nhập dữ liệu
- Lấy giá trị từ trường
- Chiến lược thiết kế hướng đối tượng
- Sử dụng thẻ CRC (classes – responsibilities – collaborations)
- Chiến lược phân rã chức năng

5.1 Nhập dữ liệu từ hộp hội thoại

- Để nhập dữ liệu chuỗi, có thể sử dụng hộp hội thoại trong gói javax.swing. Cú pháp như sau:

```
String string =
    JOptionPane.showInputDialog( null, “Prompt
    Message”, “Dialog Title”,
    JOptionPane.QUESTION_MESSAGE);
```

5.1 Nhập dữ liệu từ hộp hội thoại



5.1 Nhập dữ liệu sử dụng Frame

- *Nút lệnh:* Button (nút) là một thành phần của khung, dùng để tạo một sự kiện khi người sử dụng bấm chuột lên nó.
- *Trường nhập dữ liệu (Data Entry Field):* Là một thành phần của khung, nơi NSD có thể nhập vào một giá trị.
  - Trong GUI, sử dụng các đối tượng văn bản, ví dụ TextField để nhập dữ liệu kiểu chuỗi
  - Sau đó chuyển kiểu dữ liệu cho phù hợp với yêu cầu của chương trình.

5.1 Nhập dữ liệu sử dụng Frame

- Phương thức tạo của lớp TextField

  public TextField(int columns)

  // Tạo một TextField có số cột xác định

  public TextField(String s, int columns)

  // Tạo một TextField có số cột xác định, với văn bản cho trước là chuỗi s

- Các phương thức lớp TextField

  // Phương thức

  public String getText( )

  // Trả về văn bản của TextField

  public void setText(String s)

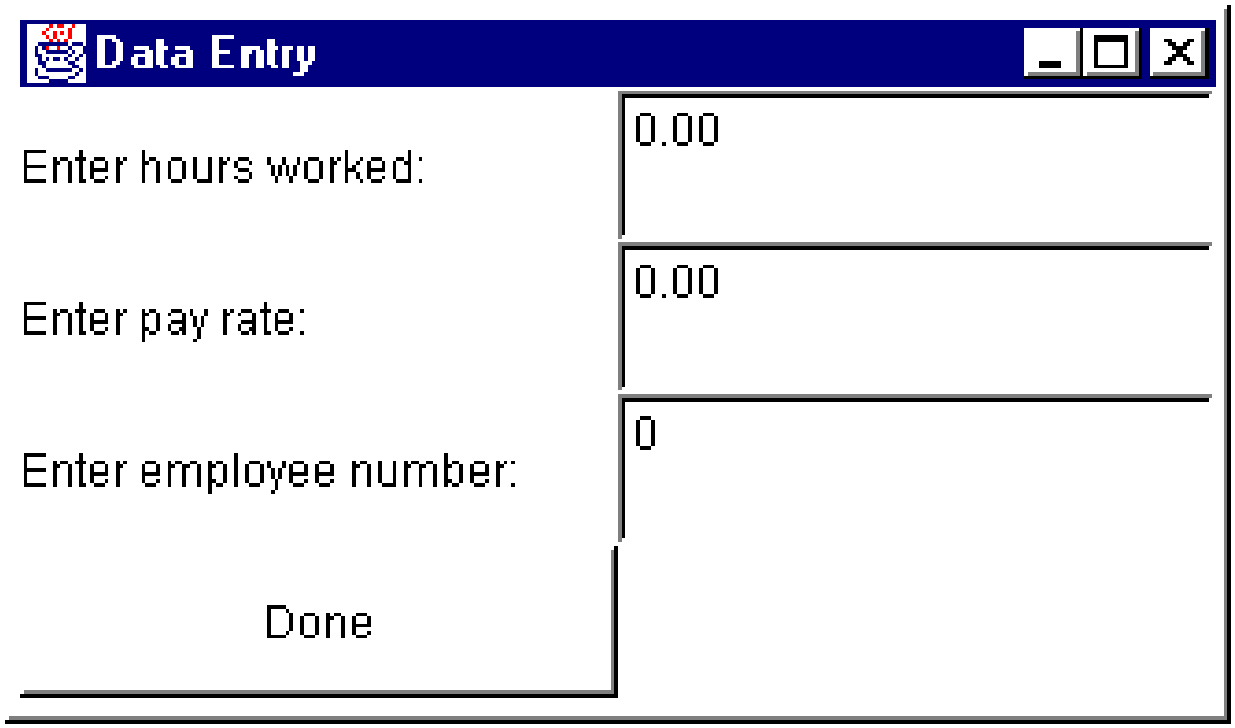
  // Thiết lập văn bản s cho TextField

5.1 Nhập dữ liệu sử dụng Frame

- *Các bước sử dụng nút lệnh (Button):*
  - Khai báo một biến Button.
  - Sử dụng *new* để tạo đối tượng Button.
  - Đặt một tên cho sự kiện Button.
  - Thêm đối tượng Button vào khung.
  - Tạo một lớp listener với *actionPerformed* để quản lý các sự kiện Button.
  - Nhận biết Listener với Button qua việc gọi phương thức *addActionListener*.

5.1 Nhập dữ liệu sử dụng Frame

- Ví dụ về khung nhập dữ liệu



5.1 Tạo một trường nhập dữ liệu

- Khai báo một biến của lớp trường tương ứng
- Tạo một đối tượng của lớp đó
- Sử dụng phương thức *add* để thêm đối tượng này vào Frame
- Ví dụ: Tạo một trường nhập dữ liệu có tên là *inputField*

```
TextField inputField; // Khai báo một trường để nhập dữ liệu
fieldLabel1 = new Label1("Enter data here:"); // Tạo nhãn Lable1
inputField = new TextField (6); // Tạo đối tượng 6 ký tự
dataEntryFrame.add(inputField); //Thêm vào khung dataEntryFrame
```

- ***Chú ý:*** Ta có thể tạo đối tượng cùng với chuỗi mặc định của nó

```
inputField = new TextField ("Replace Me",10); // Chuỗi mặc định
```

5.1 Tạo một trường nhập dữ liệu

- Ví dụ về sử dụng khung để nhập dữ liệu

```
. . .
Frame dataEntryFrame;       //Khai báo khung
Label1 fieldLabel1;         //Khai báo nhãn
TextField inputFileId;      //Khai báo trường
Button done;                //Khai báo nút
ButtonHandler myHandler;    //Khai báo đối tượng xử lý sự kiện
  . . .
dataEntryFrame = new Frame();      //Tạo khung
fieldLabel1 = new Label1("Enter data here:");          //Tạo nhãn
inputFileId = new TextField("Replace Me", 10);         //Tạo trường
done = new Button("Done");              //Tạo nút
```

5.1 Tạo một trường nhập dữ liệu

- Ví dụ về sử dụng khung để nhập dữ liệu (tt)

```
done.setActionCommand("done");      //Đặt tên cho sự kiện nút
dataEntryFrame.setLayout (new FlowLayout()); //Thiết lập bố cục
dataEntryFrame.add(fieldLabel1);    //Thêm nhãn vào khung
dataEntryFrame.add(inputFileld);    //Thêm trường vào khung
dataEntryFrame.add(done);           //Thêm nút vào khung
myHandler = new ButtonHandler( );  //Tạo listener
done.addActionListener(myHandler); //Đăng ký listener
dataEntryFrame.pack()               //Đóng gói khung
dataEntryFrame.show()               //Hiển thị khung
  . . .
```

5.4 Lấy giá trị từ trường

Chuyển chuỗi thành số thực:

```
double  price ;
TextField  priceField;

price = Double . valueOf( priceField . getText( ) ) . doubleValue( ) ;
```

Đối tượng String

Đối tượng ẩn của lớp Double

Phương thức instance

Chuyển thành số nguyên:

```
int  quantity ;
TextField  quantityField;

quantity = Integer . valueOf( quantityField . getText( ) ) . intValue( ) ;
```

Đối tượng String

Đối tượng ẩn của lớp Integer

Phương thức instance

5.4 Lấy giá trị từ trường

```
// Chương trình Rainfall
import  java.awt.*;             // Nạp gói java.awt
import  java.awt.event.*;    // Nạp gói java.awt.event
public class Rainfall {
  // Định nghĩa một button listener
  private static class ActionHandler implements ActionListener {
     public void actionPerformed(ActionEvent event)
     // Xử lý sự kiện từ nút Enter trong inputFrame  {
        double amount, double average; // Lưu giá trị vào và giá trị trung bình
        // Chuyển chuỗi vào inputField thành số thực
        amount =  Double.valueOf(inputField.getText()).doubleValue();
        numberEntries++;
        total = total + amount;
        average = total / numberEntries;
        outputLabel.setText("" + average); //Xuất dữ liệu
        inputField.setText("");                        // Xóa trường
     }
  }
```

5.4 Lấy giá trị từ trường

```
// Khai báo các biến cho lớp Rainfall
 private static Frame inputFrame;   // Khai báo khung
 private static Label outputLabel;    // Khai báo nhãn
 private static TextField inputField; // Khai báo trường
 private static double total;        // Lưu trữ tổng
 private static double numberEntries; // Đếm số đầu vào
 public static void main( String[ ] args )
 {
    Label entryLabel;              // Nhãn cho trường nhập
    Button enter;                        // Nút Enter
    ActionHandler action;          // Khai báo listener
    // Tạo và khởi tạo các biến
    total = 0.0;
    numberEntries = 0.0;
    inputFrame = new Frame( );
    entryLabel = new Label("Enter amount here:");
```

5.4 Lấy giá trị từ trường

```
outputLabel = new Label(“0.0”, Label.RIGHT);
inputField = new TextField(“”, 10);
enter = new Button(“Enter”);
enter.setActionCommand(“enter”); // Đặt tên sự kiện nút
action = new ActionHandler( );  // Tạo listener
enter.addActionListener(action); // Đăng ký listener
// Đưa các thành phần vào khung
inputFrame.setLayout(new GridLayout(2,2));
inputFrame.add(entryLabel);
inputFrame.add(inputField);
inputFrame.add(enter);
inputFrame.add(outputLabel);
inputFrame.pack( );
inputFrame.show( );
```

5.4 Lấy giá trị từ trường

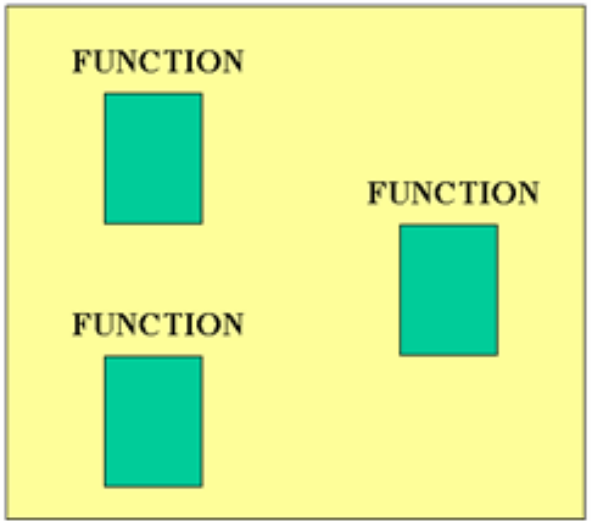
```
    inputFrame.addWindowListener( new WindowAdapter( )
    // Xử lý sự kiện đóng cửa sổ (khung)
       {
          public void windowClosing (WindowEvent event)
          {
             inputFrame.dispose( );     // Xóa khung
             System.exit( 0 );          // Thoát chương trình
          }
       });
  }
}
```

5.4 Chiến lược thiết kế HĐT (OOD)

- *Hai chiến lược thiết kế phần mềm thường dùng:* Phân rã chức năng (module hóa bài toán) và hướng đối tượng.
- *Chiến lược phân rã chức năng:*
  - Phân rã bài toán ban đầu được thành các bài toán con nhỏ hơn, dễ giải quyết hơn.
  - Các kết quả sẽ được kết hợp với nhau để tạo thành kết quả của bài toán ban đầu.
- *Chiến lược thiết kế hướng đối tượng:* Bao gồm năm giai đoạn
  - Phân tích yêu cầu (Requirement Analysist)
  - Phân tích (Analysist)
  - Thiết kế (Design)
  - Lập trình (Programming)
  - Kiểm tra (Testing)
- Lập trình hướng đối tượng (OOP – Object Oriented Programming) là giai đoạn thứ tư của thiết kế hướng đối tượng.

5.1 Chiến lược thiết kế hướng đối tượng

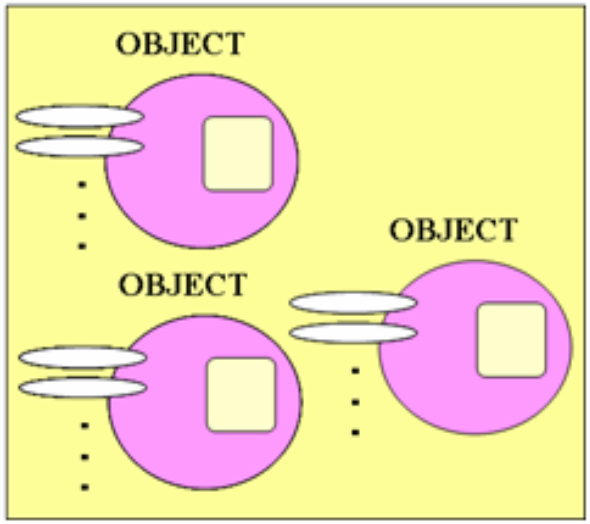
Phân rã
chức năng



Phân rã chức năng

Bài toán được phân rã thành các bài toán con dễ quản lý hơn, các kết quả sẽ được kết hợp với nhau để tạo thành kết quả của bài toán ban đầu

Thiết kế
hướng đối tượng



Thiết kế hướng đối tượng

Lời giải được biểu diễn theo thuật ngữ đối tượng (các thực thể tự chứa, gồm dữ liệu và các toán tử thực hiện trên dữ liệu đó), tương tác bằng cách gởi thông điệp qua lại lẫn nhau

5.4 Chiến lược thiết kế hướng đối tượng

- Đối tượng: Là một thực thể, chứa các giá trị xác định tính chất, và các phương thức (methods) xác định hành vi.
- Lớp: Khuôn mẫu (template) của đối tượng (Object), bao gồm dữ liệu và các phương thức tác động lên dữ liệu.
  - Các đối tượng được xây dựng bởi các lớp nên được gọi là các thể hiện của lớp (class instance).
- Thừa kế: Định nghĩa một lớp mới trên cơ sở phát triển định nghĩa của một lớp đã tồn tại để đáp ứng những yêu cầu mới.
- Siêu lớp (superclass) và lớp con (subclass):
  - Lớp được thừa kế gọi là siêu lớp.
  - Lớp thừa kế gọi là lớp con.

5.6 Sử dụng thẻ CRC

- OOD bắt đầu bằng việc xác định các lớp đối tượng chính và chọn các thuộc tính trừu tượng tương ứng cho các đối tượng đó.
- Mỗi lớp chỉ ra một tập hợp các đáp ứng, là các hành động mà các đối tượng của nó hỗ trợ.
- Các đối tượng cộng tác với nhau qua việc gửi thông điệp.

5.6 Sử dụng thẻ CRC

- Các lớp (classes), đáp ứng (responsibilities), và cộng tác (collaborations) kết hợp lại để giải quyết một vấn đề.
- Sử dụng thẻ các lớp (classes), đáp ứng (responsibilities) và cộng tác (collaborations) là một kỹ thuật để phát triển các thiết kế hướng đối tượng.
- Phương pháp này cần vận dụng trí tuệ tập thể, tính xuyên suốt và tạo kịch bản để xác định và tinh chế các lớp cần thiết cho bài toán.

5.6 Sử dụng thẻ CRC

- CRC của lớp địa chỉ

Tên lớp: Address	**Lớp cha:**	**Lớp con:**
Đáp ứng		**Cộng tác**
Tạo ra chính nó		Tên: Tên
Cho biết tên		Tên: Họ, lót, tên
Cho biết thành phố		Không
Cho biết tiểu ban		Không

5.1 Chiến lược phân rã chức năng

- Đi từ trừu tượng (danh sách các bước giải bài toán chính mà một số chi tiết còn chưa được làm rõ) đến cụ thể (các chi tiết thực hiện đều hoàn toàn rõ ràng).
- Tập trung vào dãy các hoạt động (giải thuật) cần thiết để giải quyết bài toán.
- Bắt đầu bằng việc chia lời giải thành một loạt các bước chính, tiếp tục cho đến khi mỗi bài toán con là không thể phân chia được nữa hoặc đã có một lời giải hiển nhiên.
- Chương trình là tập hợp các module giải các bài toán con.
- Một lược đồ cấu trúc module (cây lời giải phân cấp) thường được tạo ra.
- Dữ liệu đóng vai trò thứ yếu, hỗ trợ cho các hoạt động (giải thuật).

Câu hỏi và bài tập

1. Một sự kiện sẽ được gửi đi đâu khi người sử dụng bấm chuột trên một nút?
2. Điều gì xảy ra khi người sử dụng bấm chuột trên một nút?
3. Khai báo một *TextField* với tên gọi là *dataField*.
4. Gán một giá trị vào *dataField* với độ dài là 10.
5. Hãy gán một giá trị ngầm định “Replace me” vào *dataField*.
6. Thêm *dataField* vào đối tượng khung *dataFrame.*
7. Viết lệnh lưu trữ các giá trị nhập ở *dataField* vào biến *dataValue* có kiểu *String*.
8. Viết lệnh chuyển *dataValue* vào *doubleValue* kiểu *double*.
9. Khai báo một biến *press* kiểu *Button*.

Câu hỏi và bài tập

10. Viết lệnh gán tên "Press" cho *press*.
11. Khai báo một đối tượng thuộc lớp *ButtonListener*.
12. Viết lệnh đăng ký *ButtonListener* này.
13. Gọi tên ba đối tượng có thể chứa trong đối tượng *Frame*.
14. Các bước để thêm một trường nhập dữ liệu vào một đối tượng *Frame* là gì?
15. Điều gì sẽ xảy ra nếu ta quên thêm chỉ thị *dataFrame.show()*?
16. Ba bước cần thiết để thêm một đối tượng nút lệnh vào một khung là gì?
17. Kiểu của biểu thức *Integer.valueOf("1999")* là gì? Nếu nó trả về một giá trị kiểu *Integer*, làm thế nào để chuyển nó về kiểu *int*?

$radians = degrees \times \frac{\pi}{180}$

Câu hỏi và bài tập

18. Thiết kế và viết một chương trình ứng dụng để đọc một số hoá đơn, số lượng đặt hàng, và đơn giá (tất cả đều là số nguyên) rồi tính tổng giá tiền. Ứng dụng này sẽ viết ra số hoá đơn, số lượng, đơn giá, và tổng giá tiền trên một cửa sổ với các phần riêng biệt.

Chương 6
ĐIỀU KIỆN, BIỂU THỨC LOGIC VÀ CẤU TRÚC CHỌN

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 6

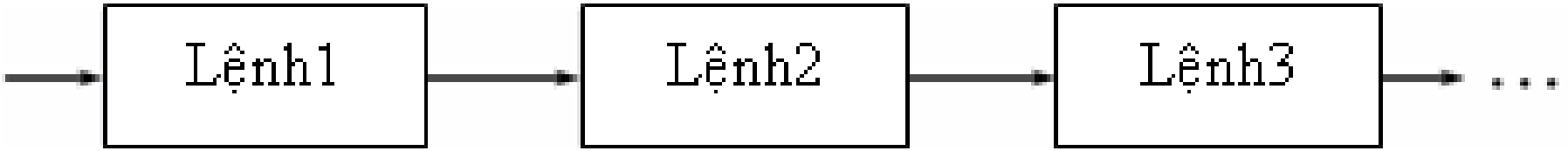
- Luồng điều khiển
- Điều kiện và biểu thức logic
- Cấu trúc if
- Cấu trúc if lồng nhau
- Cấu trúc switch
- Quản lý các sự kiện nhiều nút bấm

6.1 Luồng điều khiển

- Luồng điều khiển: trật tự mà máy tính thực hiện các lệnh trong một chương trình.
- Cấu trúc điều khiển: là một lệnh được sử dụng để làm thay đổi luồng điều khiển tuần tự một cách bình thường.
- Ngôn ngữ Java có các dạng cấu trúc điều khiển chung như sau:

6. Luồng điều khiển

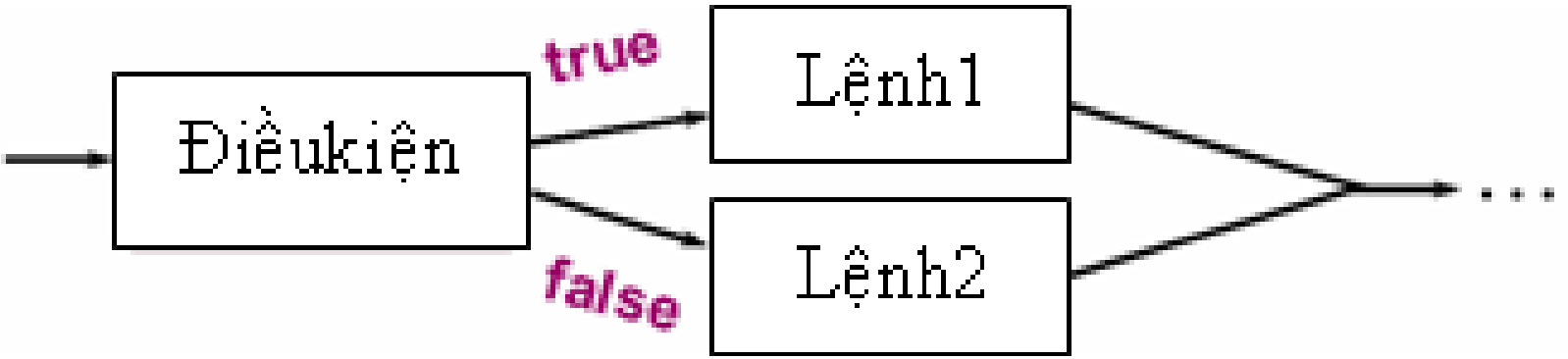
- Cấu trúc tuần tự (sequence): Một dãy các lệnh được thực hiện từ đầu đến cuối, lệnh này tiếp lệnh kia.



6. Luồng điều khiển

- Cấu trúc lựa chọn (còn gọi là cấu trúc rẽ nhánh hoặc cấu trúc quyết định): Thực hiện các lệnh khác nhau tuỳ thuộc vào các điều kiện xác định.

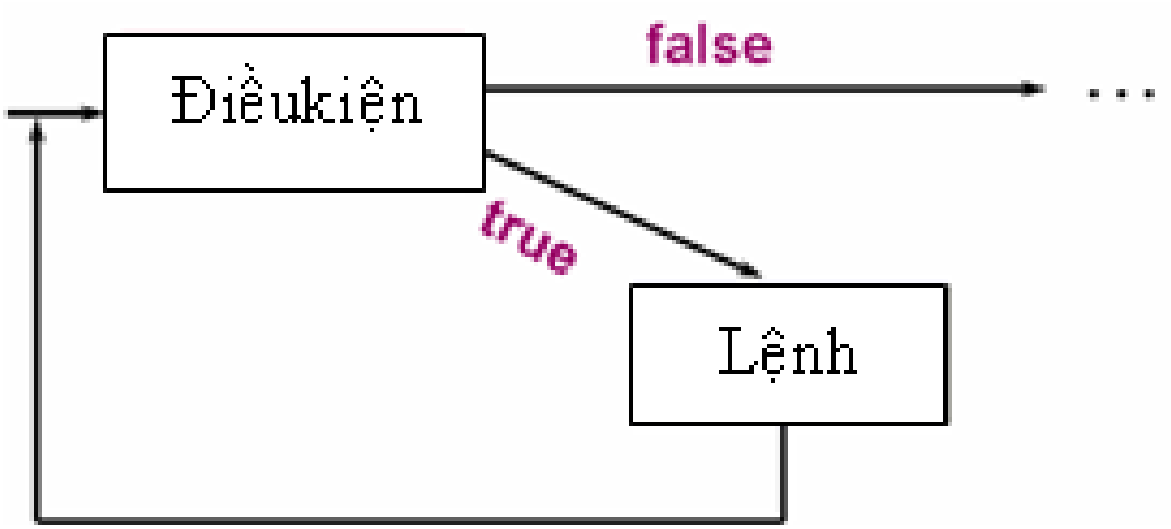
  if Điềukiện Lệnh1 else Lệnh2;



6. Luồng điều khiển

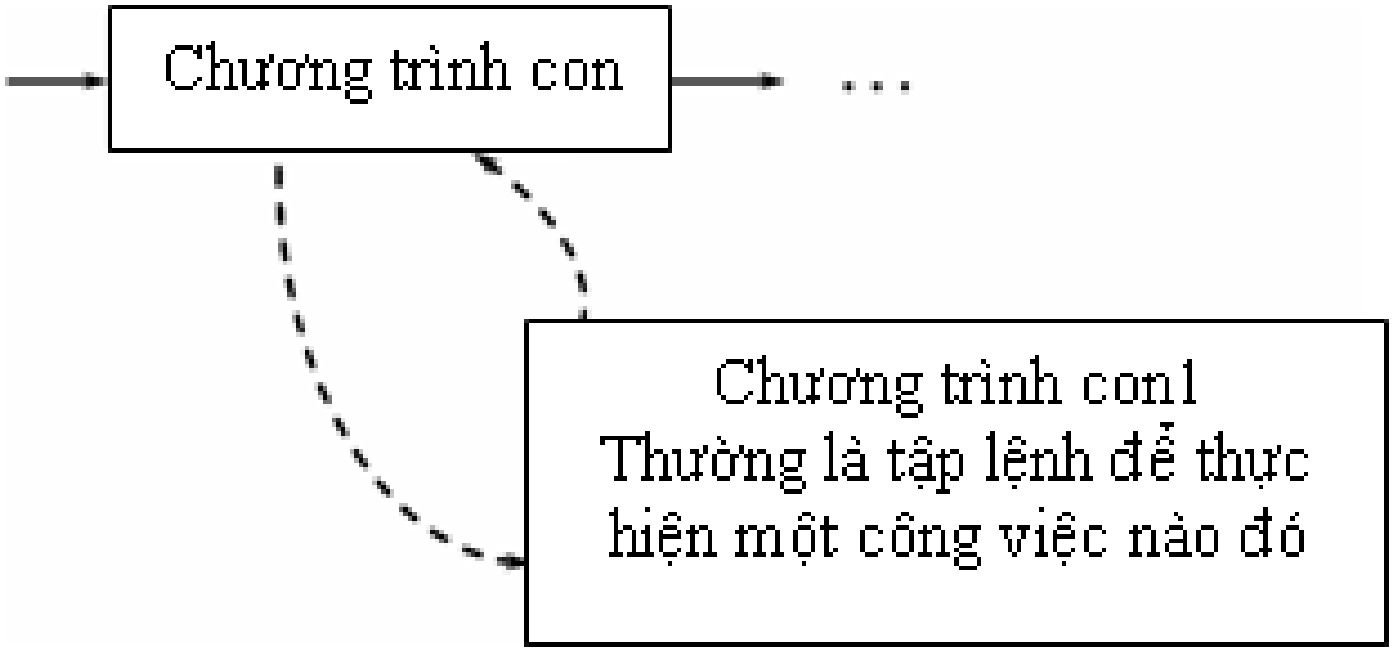
- Cấu trúc lặp (loop, repetition hoặc decision): Lặp lại các lệnh trong khi các điều kiện xác định còn đúng.

  while Điềukiện do Lệnh;



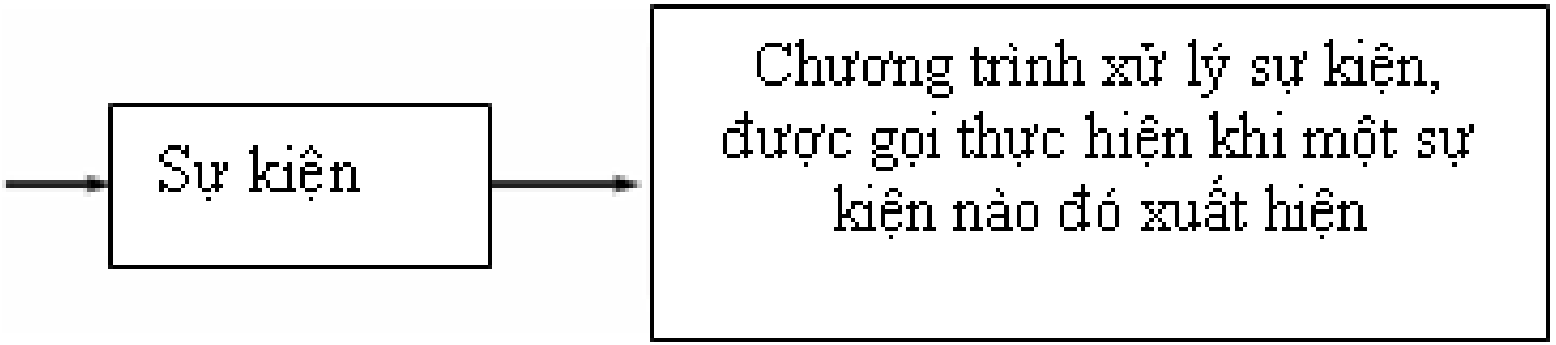
6. Luồng điều khiển

- Cấu trúc chương trình con: Một chương trình con sẽ chia chương trình chính thành các đơn vị nhỏ hơn.



6. Luồng điều khiển

- Cấu trúc không đồng bộ: Cấu trúc này xử lý các sự kiện bắt nguồn từ bên ngoài chương trình, ở thời điểm bất kỳ, ví dụ sự kiện bấm một nút.



6.1 Điều kiện và biểu thức logic

- *Kiểu dữ liệu Boolean:* là một kiểu nguyển thuỷ, chỉ bao gồm hai giá trị là các hằng **true** (T - đúng), và **false** (F - sai).
- Khai báo các biến thuộc kiểu Boolean:

```
boolean   hasFever;
boolean   isSenior;
boolean a1 = true;
boolean a2 = false;
boolean b = (1 > 2);
boolean b2 = (1 = = 2);
```

6.1 Điều kiện và biểu thức logic

- Phép so sánh: Kết quả là một giá trị logic
- Các toán tử so sánh:

Toán tử	Tên gọi	Ví dụ
<	nhỏ hơn	a < b
<=	nhỏ hơn hoặc bằng	a1 <= a2
>	lớn hơn	x > y
>=	lớn hơn hoặc bằng	x1 >= x2
==	bằng	x == y + 1
!=	không bằng	x != y

6.1 Điều kiện và biểu thức logic

- Các toán tử Boolean:

Toán tử	Tên gọi	Ví dụ
!	phủ định	!b
&&	và	(1<x) && (x<100)
||	hoặc	a1 || a2
^	hoặc loại trừ (xor)	a1 ^ a2

- Ví dụ:

bt1 && bt2

(1 < x) && (x < 100)

Nếu x bằng 1, x sẽ bằng bao nhiêu sau khi thực hiện các biểu thức sau?

(1 > x) && (1 > x++);

(1 == x) || (10 > x++);

6.1 Điều kiện và biểu thức logic

- Bảng giá trị của các toán tử

a	b	Not a	a And b	a Or b	a Xor b
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

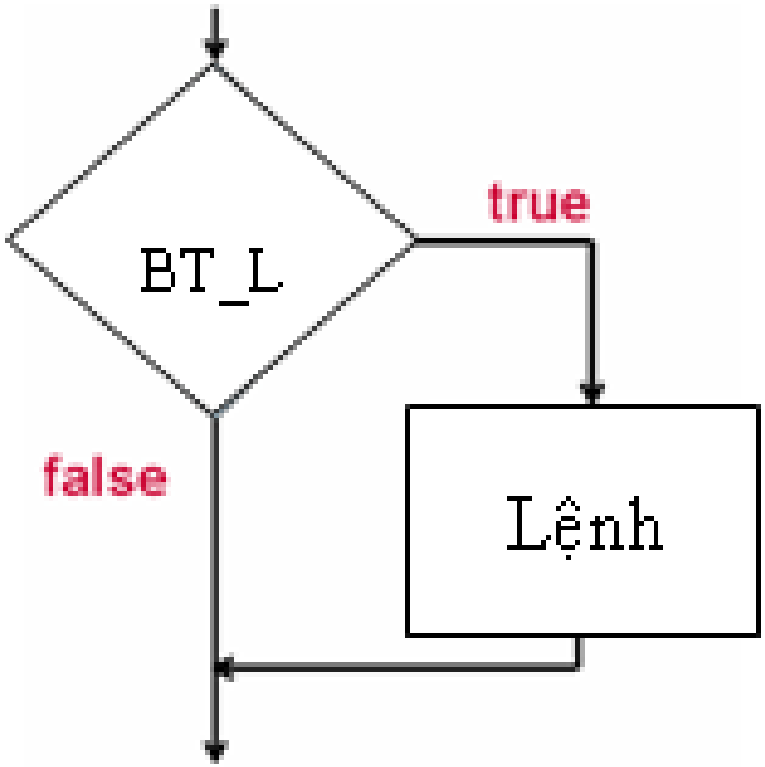
6.1 Điều kiện và biểu thức logic

- Biểu thức logic Boolean chỉ nhận giá trị *true* (đúng) hoặc *false* (sai).
- Một biểu thức logic là sự kết hợp đúng, giữa các toán tử so sánh và logic, và các toán hạng.
- Mỗi biểu thức có một giá trị là true hoặc false.

6.3 Cấu trúc if

- Cú pháp:

```
if (BT_L) {
    Lệnh;
}
```



6.3 Cấu trúc if

- Ý nghĩa: Nếu *BT_L* (là một biểu thức logic) trả về giá trị **true** thì thực hiện *Lệnh*, ngược lại, không làm gì cả.
- Ví dụ:

```
if ((i > 0) && (i < 10)) {
  System.out.println("i la nguyen giua 0 va 10");
}
```

6.3 Cấu trúc if

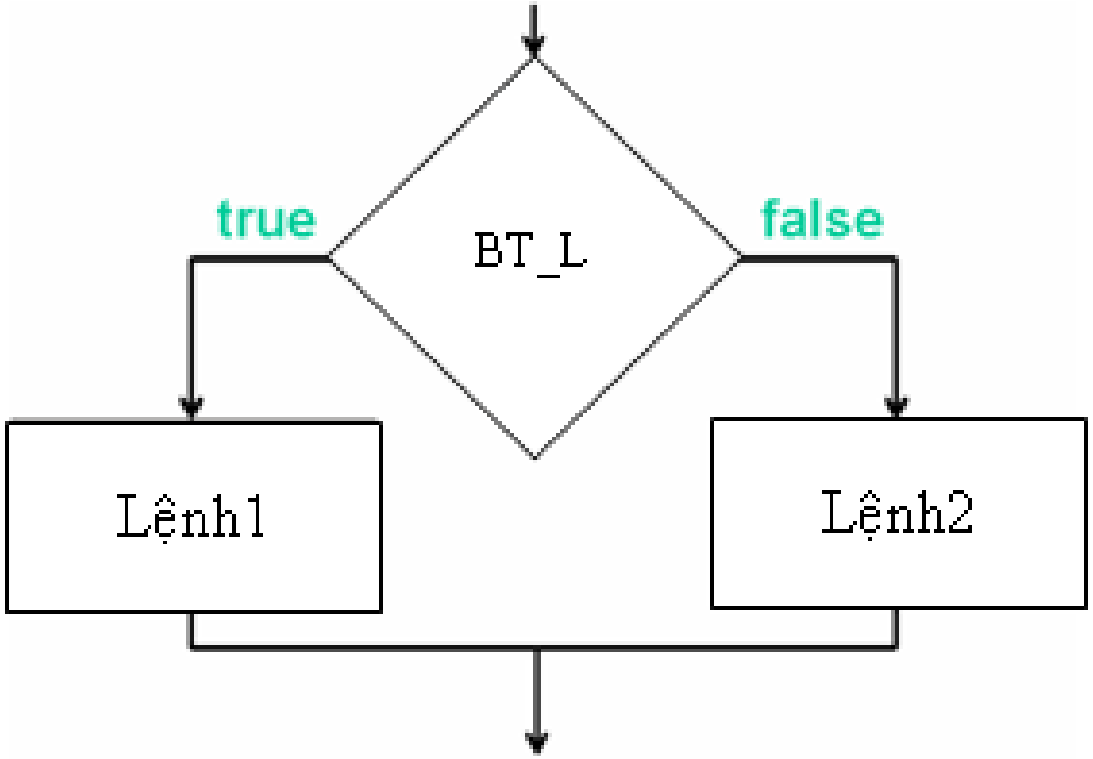
- Lỗi phổ biến: thêm một dấu chấm phẩy ở cuối mệnh đề if.

```
if (radius >= 0);
{
  area = radius*radius*PI;
  System.out.println("The area for the circle of
   radius " +
    radius + " is " + area);
}
```

6.3 Cấu trúc if

- Lệnh if...else:
- Cú pháp:

```
if (BT_L) {
  Lệnh1;
}
else {
  Lệnh2;
}
```



6.3 Cấu trúc if

- Ý nghĩa: Nếu *BT_L* trả về giá trị **true** thì thực hiện *Lệnh1*, ngược lại, thực hiện *Lệnh2*.
- Ví dụ:

```
if (bankinh >= 0) {
   dientich = bankinh*bankinh*PI;
   System.out.println("Dien tich hinh tron co ban
   kinh " + bankinh + " la " + dientich);
}
else {
  System.out.println("Du lieu khong hop le!");
}
```

6.4 Cấu trúc if lồng nhau

- Cú pháp:

```
if ( BT_L1 )
                Lệnh1;

    else if ( BT_L2 )
                Lệnh2
                .
                .
                .
    else if ( BT_Ln )
                Lệnhn
    else
                Lệnhn+1;
```

6.4 Cấu trúc if lồng nhau

- Ý nghĩa: Thực hiện $Lệnh_i$, (i = 1 .. n) ứng với BT_L_i trả về giá trị *true* đầu tiên. Ngược lại, nếu không có BT_L_i nào đúng thì lệnh $lệnh_{n+1}$ sẽ được thực hiện.
- Chú ý:
  - Chỉ có một phát biểu duy nhất được thực hiện
  - Mệnh đề *else* gắn với mệnh đề *if* gần nhất trong cùng một khối.

6.4 Cấu trúc if lồng nhau

- Ví dụ, đoạn lệnh sau

```
int i = 1; int j = 2; int k = 3;
if (i > j)
        if (i > k)
                System.out.println("A");
        else
                System.out.println("B");
```

- là tương đương với

```
int i = 1; int j = 2; int k = 3;
if (i > j) {
        if (i > k)
                System.out.println("A");
        else
                System.out.println("B");}
```

6.4 Cấu trúc if lồng nhau

- Đoạn lệnh trước sẽ không in ra gì cả.
- Muốn mệnh đề *else* gắn với mệnh đề *if* đầu tiên, ta phải thêm một cặp ngoặc nhọn:

```
int i = 1; int j = 2;int k = 3;
  if (i > j) {
    if (i > k)
      System.out.println("A");
  }
  else
    System.out.println("B");
```

6.5 Cấu trúc switch

- Cú pháp:

```
switch (bt_switch) {
  case gtri1: lệnh1;
  case gtri2: lệnh2;
   ……
  case gtrin: lệnhn;
  default: lệnhn+1;
}
```

6.5 Cấu trúc switch

- Ý nghĩa:
  - Nếu giá trị của một biến $gtri_i$ nào đó bằng với giá trị của bt_switch thì các $lệnh_i$ trở đi (cho đến cuối cấu trúc) được thực hiện (i = 1..n).
  - Ngược lại, nếu không có $gtri_i$ nào bằng với giá trị của bt_switch thì lệnh $lệnh_{n+1}$ (sau default, nếu có) sẽ được thực hiện.

6.5 Cấu trúc switch

- Ví dụ:

```
switch (sonam) {
  case 7:  laisuatnam = 7.25;
  case 15: laisuatnam = 8.50;
  case 30: laisuatnam = 9.0;
  default: System.out.println ("Sai so nam, nhap 7,
   15, hoac 30");
}
```

6.5 Cấu trúc switch

- Quy tắc sử dụng lệnh switch
  - Biểu thức bt_switch phải sinh ra một giá trị kiểu **char**, **byte**, **short**, hoặc **int**, và phải luôn được bao trong cặp dấu ngoặc tròn.
  - $gtri_1$, ..., $gtri_n$ phải có cùng kiểu dữ liệu với giá trị của bt_switch.
  - Nên sử dụng từ khóa **break** cuối mỗi trường hợp để thoát khỏi phần còn lại của lệnh **switch**. Nếu không có lệnh **break**, lệnh **case** tiếp theo sẽ được thực hiện.

6.5 Cấu trúc switch

- Quy tắc sử dụng lệnh switch (tt)
  - Trường hợp **default** là tùy chọn, có thể sử dụng để thực hiện các lệnh khi không có trường hợp nào ở trên là đúng.
  - Thứ tự của các trường hợp (gồm cả trường hợp **default**) là không quan trọng. Tuy nhiên, phong cách lập trình tốt là nên theo một trình tự logic của các trường hợp và đặt trường hợp **default** ở vị trí cuối cùng.

6.5 Cấu trúc switch

- Lưu ý: Đừng quên dùng lệnh **break** khi cần thiết. Ví dụ đoạn mã sau

```
switch (sonam) {
  case 7:  laisuatnam = 7.25;
  case 15: laisuatnam = 8.50;
  case 30: laisuatnam = 9.0;
  default: System.out.println("Sai so nam!");
}
```

6.6 Quản lý các sự kiện nhiều nút bấm

- Sau khi tạo ra một nút với phương thức *new*, gọi *setActionCommand* với tham số *String* để xác định nút này.
- Nếu không, listener sẽ giả sử rằng tên của nút cũng giống như chuỗi được sử dụng khi tạo đối tượng nút.
- Sau đó gọi *addActionListener*.
- Ví dụ, xây dựng một giao diện người dùng với hai nút bấm, *Copy* và *Done*

6.6 Quản lý các sự kiện nhiều nút bấm

buttonAction = new ButtonHandler(); //Tạo đối tượng ButtonHandler

copy = new Button(“Copy”); //Tạo đối tượng nút với chuỗi Copy

copy.setActionCommand(“copy”); //Gán một chuỗi cho nút để định danh

copy.addActionListener(buttonAction); //Đăng ký button listener

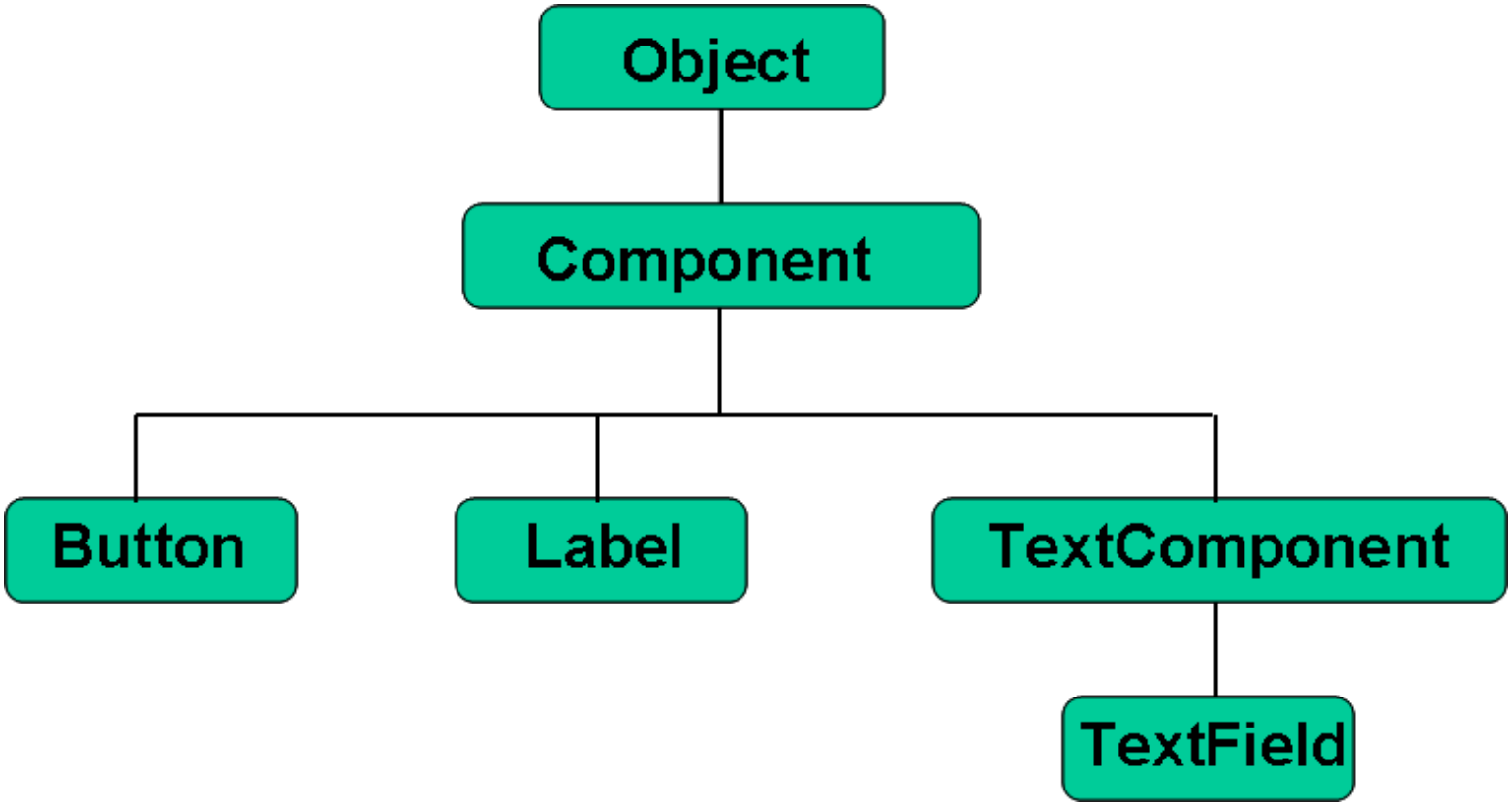
done = new Button(“Done”); //Tạo đối tượng nút với chuỗi Done

done.setActionCommand(“done”); //Gán một chuỗi cho nút để định danh

done.addActionListener(buttonAction); //Đăng ký button listener

6.6 Quản lý các sự kiện nhiều nút bấm

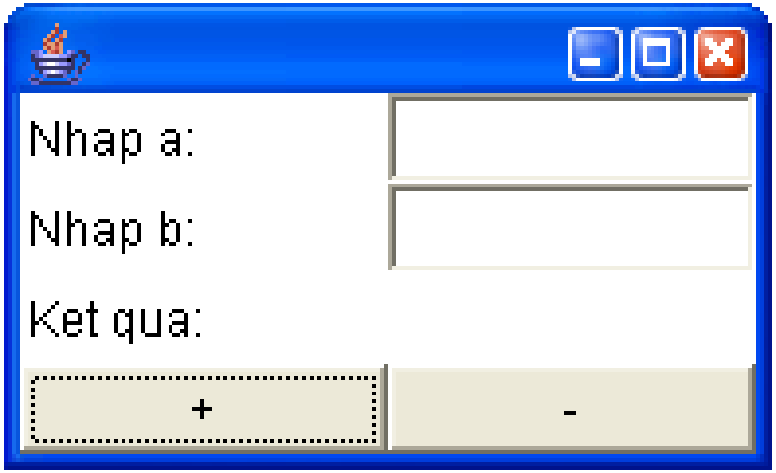
- Ví dụ: Viết chương trình mô phỏng máy tính (calculator)



6.6 Quản lý các sự kiện nhiều nút bấm

- Thẻ CRC của máy tính:

Tên lớp: Calculator	Lớp cha:	Lớp con:
Đáp ứng		**Cộng tác**
Tạo cửa sổ nhập		Trường văn bản, nhãn, chuỗi, nút bấm
Xử lý sự kiện nút số		Nút bấm
Xử lý sự kiện nút xóa		Nút bấm
Xử lý sự kiện đóng cửa sổ		Khung



6.6 Quản lý các sự kiện nhiều nút bấm

```
import  java.awt.*; import  java.awt.event.*;
public class CongTru{
  private static class ActionHandler implements ActionListener{
     public void actionPerformed(ActionEvent event){
        double a,b,c;
        a =  Double.valueOf(inputField.getText()).doubleValue();
        b =  Double.valueOf(inputField1.getText()).doubleValue();
        String s = event.getActionCommand();
        if (s.equals("cong")) c = a + b;
        else c = a - b;
        outputLabel.setText("" + c);
        // Clear input field
        inputField.setText("");inputField1.setText("");
     }
  }
```

6.6 Quản lý các sự kiện nhiều nút bấm

```
private static Frame inputFrame;   // User interface frame
  private static Label outputLabel;
  private static TextField inputField, inputField1;
  public static void main( String[ ] args ){
     Button cong, tru;                  // Declare buttons
     ActionHandler action;              // Declare listener
     inputFrame = new Frame( );
     inputFrame.setLayout(new GridLayout(4,2));
     inputFrame.add(new Label("Nhap a:"));
     inputField = new TextField("", 10);
     inputFrame.add(inputField);
     inputFrame.add(new Label("Nhap b:"));
     inputField1 = new TextField("", 10);
     inputFrame.add(inputField1);
     inputFrame.add(new Label("Ket qua:"));
     outputLabel = new Label("");
     inputFrame.add(outputLabel);
```

6.6 Quản lý các sự kiện nhiều nút bấm

```
    cong = new Button("+");
    cong.setActionCommand("cong"); // Name button event
    action = new ActionHandler( );  // Instantiate listener
    cong.addActionListener(action); // Register listener
    inputFrame.add(cong);                          // Add button
    tru = new Button("-");
    tru.setActionCommand("tru"); // Name button event
    tru.addActionListener(action); // Register listener
    inputFrame.add(tru);                  // Add button
    inputFrame.pack( );
    inputFrame.show( );
    inputFrame.addWindowListener( new WindowAdapter( ){
          public void windowClosing (WindowEvent event){
            inputFrame.dispose( );     // Remove frame
            System.exit( 0 );        // Quit program
          }
        });
  }
}
```

Chương 7

LỚP VÀ PHƯƠNG THỨC

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 7

- Đóng gói
- Thiết kế giao diện lớp
- Mô tả dữ liệu nội bộ
- Cú pháp khai báo lớp
- Khai báo phương thức
- Gói

7.1 Đóng gói

- Chi tiết thi hành lớp là ẩn đối với người sử dụng lớp → đóng gói (encapsulation).
- Đóng gói là sự thiết kế một lớp sao cho sự thực hiện của nó được bảo vệ khỏi tác động của các mã ngoài, trừ khi qua một giao diện hình thức (formal interface).
- Các phương thức dùng chung (public methods) của một lớp cung cấp giao diện giữa mã ứng dụng và các đối tượng lớp.
- Đóng gói giúp giấu đi các chi tiết cài đặt và dữ liệu cục bộ
- Công bố ra ngoài những gì cần thiết để trao đổi với các đối tượng khác.
- Đơn vị đóng gói cơ bản là lớp (class).
- Lớp định rõ những thành phần dữ liệu và các đoạn mã cài đặt các thao tác xử lý trên các đối tượng dữ liệu đó.

7.1 Đóng gói

- Các ưu điểm của đóng gói:
  - Bảo vệ nội dung của lớp khỏi bị hư hỏng do tác dụng của mã ngoài
  - Đơn giản hoá việc thiết kế các chương trình lớn bằng cách phát triển các phần riêng biệt với nhau
  - Cho phép thay đổi việc thực thi các lớp sau khi đã tạo và phát triển chúng
  - Cho phép tái sử dụng một lớp từ các ứng dụng khác, và mở rộng lớp này để tạo nên các lớp mới có liên quan

7.1 Đóng gói

- Trừu tượng: trừu tượng dữ liệu và trừu tượng điều khiển.
  - Trừu tượng dữ liệu: sự tách biệt biểu diễn logic của một miền giá trị từ sự sử dụng.
  - Trừu tượng điều khiển: sự tách biệt thuộc tính logic của các phương thức trên một đối tượng từ sự thực hiện.
- Trạng thái đối tượng: tập các giá trị hiện tại mà nó chứa.

7.1 Đóng gói

- Đối tượng được biểu diễn bởi trạng thái khởi tạo.
- Đối tượng có thể thay đổi trạng thái của nó được gọi là có thể thay đổi (mutable).
- Đối tượng không thể thay đổi trạng thái một khi đã được tạo ra gọi là đối tượng không thể thay đổi (immutable).
- Bộ biến đổi (transformer): Phương thức làm thay đổi trạng thái của một đối tượng

7.1 Thiết kế giao diện lớp

- Các đối tượng được xây dựng bởi các lớp → các thể hiện của lớp (class instance).
- Ví dụ lớp Name:

Tên lớp: Name	Lớp cha:	Lớp con:
Đáp ứng		Cộng tác
Tạo chính nó (Họ, họ lót, tên)		Không
Cho biết họ		Không
Cho biết họ lót		Không
Cho biết tên		Không
Hai tên bằng nhau?		String
So sánh hai tên		String

7.1 Thiết kế giao diện lớp

- Chuyển thẻ CRC thành một thiết kế: chuyển các đáp ứng từ dạng mệnh đề hoặc câu thành các tên để biểu diễn đáp ứng.
- Thiết kế giao diện chung: Có hai cách
  - Các giá trị trường chung: Lưu trữ mỗi phần của tên trong một chuỗi được khai báo như một trường, ví dụ

    ```
    String first;
    String middle;
    String last;
    ```

  - Các đáp ứng là phương thức: Mỗi đáp ứng có thể được thực hiện bởi một phương thức trong lớp.

7.1 Mô tả dữ liệu nội bộ

- Đầu tiên: quyết định việc biểu diễn bên trong của dữ liệu.
- Phương thức thực hiện: Các đáp ứng của thẻ CRC là các phương thức thực hiện.
- Đáp ứng tham chiếu đến một đối tượng sẽ được thực hiện như một phương thức cụ thể (thể hiện), phải được gọi thông qua tên của đối tượng (không gọi thông qua tên lớp), được khai báo không có từ khóa static.
- Các dạng dữ liệu: dữ liệu thể hiện, dữ liệu lớp và dữ liệu cục bộ.
  - Dữ liệu thể hiện: các biểu diễn bên trong của một đối tượng cụ thể, biểu diễn trạng thái của đối tượng, được khai báo không có từ khóa static. Ví dụ:

```
public class Name{
    String first;
    String middle;
    String last; . . .
    }
```

7.1 Mô tả dữ liệu nội bộ

- Dữ liệu lớp (class data): Có thể truy cập đối với tất cả các đối tượng của một lớp, được khai báo với từ khóa static. Ví dụ:

```
public class Name{
    // Class constant
    static final String PUNCT = “, ”; . . .
    }
```

- Dữ liệu cục bộ (local data): chỉ có thể truy cập trong khối lệnh nó khai báo. Ví dụ:

```
public int compareTo(Name otherName)
    {
      int result;    // Biến cục bộ
      . . .
      return  result;
    }
```

7.1 Mô tả dữ liệu nội bộ

- Thời gian sống của biến, hằng, đối tượng: phần thời gian thực hiện một ứng dụng, khi chúng thực sự được gán một vị trí trong bộ nhớ.
  - Thời gian sống của đối tượng: Kể từ khi đối tượng được tạo ra (phương thức *new*), JVM cung cấp không gian nhớ cho nó từ một vùng gọi là vùng tự do (free pool), hoặc heap; cho đến khi JVM xác định không có biến nào tham chiếu đến nó và giải phóng (trả lại cho heap).
- Bộ gom rác của JVM (JVM's Garbage Collector): Java chứa một bộ gom rác tự động. Một cách có chu kỳ, nó tìm và trả về cho heap mọi đối tượng không được tham chiếu

7.4 Cú pháp khai báo lớp

- Khai báo:
- Một lớp bao gồm một phần đầu (heading): khai báo gói và nạp gói nếu có.
- Tiếp theo là một khối chứa các khai báo lớp
- Bao gồm khai báo tên lớp và phạm vi truy cập
- Khai báo dữ liệu
- Khai báo phương thức

7.4 Cú pháp khai báo lớp

- Ví dụ: khai báo lớp Circle (hình tròn) như sau

```
class Circle {
  double radius = 1.0;
  Circle(){
    }

    Circle(double newRadius){
      radius = newRadius;
    }

    double findArea(){
    return radius * radius * 3.14159;
  }
}
```

7.4 Cú pháp khai báo lớp

- Khai báo biến tham chiếu đối tượng:ClassName objectReference;
  - Ví dụ:

    Circle myCircle;//myCircle là một biến tham chiếu đối tượng của lớp Circle.
- Tạo đối tượng: objectReference = new ClassName();
  - Ví dụ:

    myCircle = new Circle();//Tham chiếu đối tượng sẽ được gán cho biến myCircle.
- Khai báo/Tạo đối tượng trong một lệnh:

  ClassName objectReference = new ClassName();
  - Ví dụ:

    Circle myCircle = new Circle();

7.4 Cú pháp khai báo lớp

- Truy cập đối tượng:
  - Tham chiếu dữ liệu của đối tượng:
    - objectReference.data
    - vd: myCircle.radius
  - Gọi phương thức của đối tượng:
    - objectReference.method
    - vd: myCircle.findArea()

7.4 Khai báo phương thức

- Constructor (phương thức tạo): là một dạng đặc biệt của phương thức, được gọi để xây dựng đối tượng.
- Một constructor không có tham số được gọi là default constructor.
- Các constructor phải có cùng tên với tên lớp của nó.
- Các constructor không có kiểu dữ liệu trả về, kể cả kiểu void.
- Nó được gọi sử dụng toán tử *new* khi tạo một đối tượng và đóng vai trò tạo đối tượng. Ví dụ:

```
Circle(double r) {
  radius = r;
}
Circle() {
  radius = 1.0;
}
myCircle = new Circle(5.0);
```

7.4 Khai báo phương thức

- **Cú pháp khai báo phương thức:**

  <từ bổ nghĩa> <kiểu trả về> <Tên phương thức>
  (<danh sách đối số>){

  <khối lệnh>;

  }

- Từ bổ nghĩa: xác định phạm vi truy cập của các đối tượng khác đối với các phương thức của lớp

7.4 Khai báo phương thức

- **public**: phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- **protected**: có thể truy cập được từ lớp khai báo và những lớp dẫn xuất từ nó.
- **private**: chỉ được truy cập bên trong bản thân lớp khai báo.
- **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có nghĩa là phương thức đó có thể được thực hiện kể cả khi không có đối tượng của lớp chứa phương thức đó.
- **final**: không được khai báo chồng ở các lớp dẫn xuất.
- **abstract**: phương thức không cần cài đặt (không có phần thân), sẽ được hiện thực trong các lớp dẫn xuất từ lớp này.
- **synchoronized**: ngăn các tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa, được sử dụng trong lập trình đa tuyến.

7.4 Khai báo phương thức

- <kiểu trả về>: có thể là kiểu void, kiểu cơ sở hay một lớp.
- <Tên phương thức>: đặt theo qui ước giống tên biến.
- <danh sách thông số>: có thể rỗng
- Lưu ý:
  - Các phương thức nên được khai báo dùng từ khóa public,
  - Dữ liệu thường là dùng tiền tố private vì mục đích an toàn.
  - Các biến cục bộ (local) nên được khởi tạo sau khi khai báo.

7.4 Khai báo phương thức

- Ví dụ về khai báo phương thức:

```
public class XeMay{
    public String nhasx;
    public String model;
    private float chiphisx;
    protected int thoigiansx;
    // so luong so cua xe may: 3, 4 so
    protected int so;
    // là biến tĩnh có giá trị là 2 trong tất cả
    // các thể hiện tạo ra từ lớp xemay
    public static int sobanhxe = 2;
    //Khai báo phương thức
    public float tinhgiaban(){
      return 1.5 * chiphisx;
    }
}
```

7.4 Khai báo phương thức

- Gọi phương thức: tên của phương thức, theo sau là cặp dấu ngoặc tròn, bao quanh danh sách các đối số. Ví dụ:

```
System.out.println( "Done" );
a = Math.max( y, z );
value = Math.random( );
```

- Một lời gọi phương thức sẽ tạm thời chuyển điều khiển cho phương thức được gọi để thực hiện nhiệm vụ của nó
- Cú pháp gọi phương thức:
  - tên_phương_thức(danh sách đối số)
  - Danh sách đối số được sử dụng để truyền các giá trị đến phương thức.
  - Danh sách đối số có thể có 0, 1 hoặc nhiều thông tin, tách biệt bởi dấu phẩy.

7.4 Khai báo phương thức

- Tham số trong Java: Với kiểu dữ liệu nguyên thuỷ (**int**, **double**, **boolean …**), tham số nhận một bản sao của giá trị của đối số.
  - Các tác vụ thực hiện trên tham số không ảnh hưởng đến đối số.
- Với kiểu dữ liệu tham chiếu (như chuỗi hoặc lớp), tham số nhận một bản sao của địa chỉ, nơi đối tượng được lưu trữ.
  - Những thay đổi được tạo ra trên các thuộc tính của đối tượng tham chiếu đến bởi tham số sẽ ảnh hưởng đến đối tượng đối số.
  - Là một thực tế lập trình không tốt

7.8 Gói

- Gói bao gồm một hoặc nhiều lớp có cùng đặc tính chung nào đó.
- Cú pháp khai báo gói:

  package tên_gói;

  khai báo nạp gói;

  khai báo lớp;

7.8 Gói

- Ví dụ: Tạo gói và lưu vào thư mục con *name* với tên *Name.java*, dịch *javac name\Name.java*.

```
// Tạo lớp Name được lưu trữ trong gói name

package name;
public class Name {
  // Hằng lớp
  static final String PUNCT = “, ”;        // để định dạng
  // Khai báo Biến thể hiện
  String first; …
  // Khai báo các phương thức ….
}
```

7.8 Gói

- Để sử dụng gói: Tạo file và lưu với tên *NameDriver.java*

```
// Sử dụng gói name vừa tạo
import  name.*;
public class NameDriver{
  static Name testName; // Đối tượng Name để kiểm tra
  public static void main (String[ ] args) {
    //Khai báo thân phương thức chính
   ...
  }
}
```

Câu hỏi và bài tập

1. Trừu tượng hoá là gì?
2. Mục đích của constructor của một lớp là gì?
3. Sự khác nhau giữa kiểu nguyên thuỷ và kiểu tham chiếu?
4. Một đối tượng có thể có đối tượng khác làm thành phần được không?
5. Viết phần heading của constructor copy của lớp *AddressLabel* mà nó thay đổi trường *address*.
6. Viết phần heading của một constructor của lớp *AddressLabel* mà nó có bốn biến *String* là *name, address, city,* và *state*, và một biến *long* là *zipCode*. Trường *country* được thiết lập mặc định là "*United States*".

Chương 8

THỪA KẾ, ĐA HÌNH VÀ PHẠM VI

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 8

- Thừa kế
- Biến this và quá tải phương thức
- Tính đa hình
- Lớp Object
- Cú pháp lớp gốc
- Phạm vi truy cập
- Thực hiện một lớp gốc
- Phương thức tạo sao chép

8.1 Thừa kế

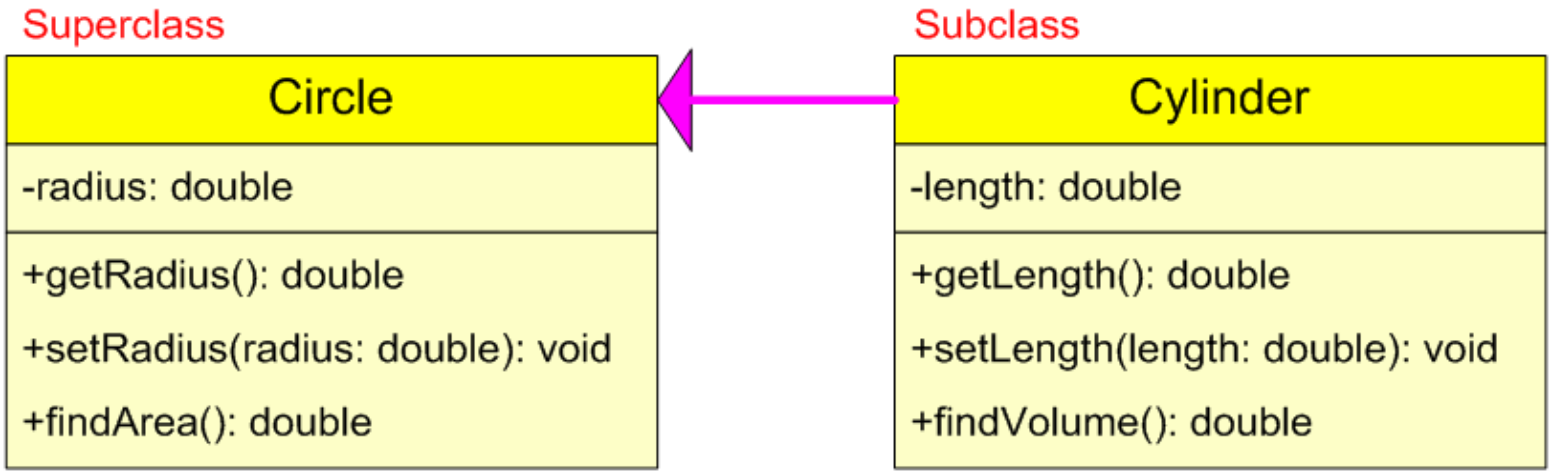
- Phát triển những lớp mới từ các lớp đã tồn tại.
- Lớp con có thể thừa kế tất cả những vùng dữ liệu và phương thức của lớp cha.
- Dùng từ khóa **extends** để chỉ lớp con.
- Ví dụ: lớp C2 (lớp các hình vuông) được phát triển từ lớp C1 (lớp các hình chữ nhật)

```
class C2 extends C1{
    Khai báo dữ liệu và phương thức của C2
}
```

- C2 được gọi là lớp con (subclass, extended class, derived class)
- C1 được gọi là lớp cha (superclass, parent class, base class)

8.1 Thừa kế

- Subclass thừa kế từ superclass các trường dữ liệu và phương thức có thể truy cập được
- Có thể thêm vào các trường dữ liệu và phương thức mới.
- Thực tế, subclass thường được mở rộng để chứa nhiều thông tin chi tiết và nhiều chức năng hơn
- Ví dụ lớp Cylinder thừa kế từ lớp Circle:



8.2 iến this và quá tải phương thức

- Là một biến ẩn tồn tại trong tất cả các lớp, được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.
- Ví dụ:

```
<tiền tố> class A {
    <tiền tố> int <field_1>;
    <tiền tố> String <field_2>;
    // Contructor của lớp A
    public A(int par_1, String par_2){
      this.field_1 = par_1;
      this.field_2 = par_2;
    }
    <tiền tố> <kiểu trả về> <method_1>(){
      // …
    }
    <tiền tố> <kiểu trả về> <method_2>(){
      this.method_1()
      // …
    }
}
```

8.2 iến this và quá tải phương thức

- Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo quá tải phương thức (overloading method).
- Ví dụ:

```
public class Xemay {
   // khai báo fields …
   public float tinhgiaban(){
            return 2 * chiphisx;
   }
   public float tinhgiaban(float huehong){
            return (2 * chiphisx + huehong);
   }
}
```

8.3 Tính đa hình

- Khả năng của ngôn ngữ cho phép đặt trùng tên phương thức và cho phép xác định phương thức thích hợp nào được gọi phụ thuộc vào lớp của đối tượng.
- Ví dụ: Định nghĩa hai đối tượng "hinh_vuong" và "hinh_tron" thì có một phương thức chung là "chu_vi". Khi gọi phương thức này, nếu đối tượng là "hinh_vuong" nó sẽ tính theo công thức khác với khi đối tượng là "hinh_tron".

8.3 ính đa hình

- Ví dụ:

```
class A_Object {
    // …
    void method_1(){
        // …
    }
}
class B_Object extends A_Object {
    // …
    void method_1(){
        // …
    }
}
```

8.3 Tính đa hình

```
class C {
        public static void main(String[] args){
                A_Object arr_Object = new A_Object[2];
                B_Object var_1 = new B_Object();
                arr_Object[0] = var_1;
                A_Object var_2;
                for (int i=0; i<2; i++){
                        var_2 = arr_Object[i];
                        var_2.method_1();
                }
        }
}
```

- i = 0, var_2 có kiểu là B_Object, var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp B_Object; i = 1, var_2 có kiểu là A_Object, var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp A_Object.
- Biến đối tượng kiểu A_Object như var_2 có thể tham chiếu đến bất kỳ đối tượng nào của bất kỳ lớp con nào của lớp A_Object. Ngược lại một biến của lớp con không thể tham chiếu đến bất kỳ đối tượng nào của lớp cha.

8.3 ính đa hình

- Từ khóa super: được dùng để thay cho superclass
- Dùng super để gọi một contructor của superclass, hoặc gọi một phương thức của superclass. Ví dụ gọi Superclass Contructor:

  super(), hoặc super(tham_số)
- Lệnh trên phải được đặt tại dòng đầu tiên của subclass constructor và là cách duy nhất để gọi một superclass constructor.

```
public Cylinder() {
}
```

=

```
public Cylinder() {
    super();
}
```

- Gọi phương thức của Superclass: super.methodName(tham_số), Ví dụ:

```
double findVolume() {
    return super.findArea() * length;
}
```

8.3 Tính đa hình

- Chồng phương thức (Overriding Method): subclass thay đổi sự thực hiện của phương thức trong superclass
- Ví dụ, *findArea* của lớp *Circle* nên được chồng trong lớp *Cylinder* để tính diện tích bề mặt hình trụ.
- Phương thức trong subclass phải có cùng signature (tên phương thức, số lượng và kiểu của các tham số theo thứ tự cho trước của chúng) và cùng kiểu dữ liệu trả về với phương thức trong superclass (Overloading method có cùng tên, nhưng phải khác signature).

8.3 Tính đa hình

- Một phương thức chỉ có thể được chồng khi nó có thể truy cập được → không thể chồng một phương thức riêng (private method).
- Một phương thức tĩnh (static method) có thể được kế thừa, nhưng không thể được chồng.
- Khi chồng, nếu phương thức trong superclass là *protected* thì có thể thay đổi phương thức chồng trong subclass thành *public*. Nếu phương thức trong superclass là *public* thì phương thức chồng trong subclass bắt buộc cũng phải là *public*.
- Ẩn trường dữ liệu (hidding data field): Ẩn trường dữ liệu của lớp cha là việc một trường dữ liệu của lớp con được đặt cùng tên với một trường dữ liệu của lớp cha.

8.4 ớp Object

- Mọi lớp trong Java đều được thừa kế từ lớp java.lang.Object. Nếu không có sự kế thừa nào được xác định khi một lớp được tạo thì superclass của nó là lớp Object. Các phương thức của lớp Object thường được sử dụng là:

  public boolean equals(Object obj)

  public int hashCode()

  public String toString()

- Phương thức equals: So sánh bằng nhau về mặt tham chiếu, trả về **true** nếu hai biến cùng tham chiếu đến một đối tượng.
- object1.equals(object2); //So sánh bằng nhau theo tham chiếu

8.5 ú pháp lớp gốc

- Đa hình cho phép các phương thức được sử dụng chung cho một dải rộng các tham số đối tượng.
- Nên lập trình theo cách dùng chung: khai báo một biến có kiểu superclass, nó sẽ có thể chấp nhận một giá trị của bất kỳ kiểu subclass nào.
- Ép kiểu đối tượng, ví dụ

  m(new Student());

  thực hiện gán đối tượng tạo bởi new Student() cho một tham số kiểu Object, tương đương với hai lệnh:

  Object obj = new Student(); // ép kiểu ngầm

  m(obj);
- Muốn ấn định obj (kiểu Object) là một đối tượng Student:

  Student std = (Student) obj; //ép kiểu tường minh

  Không viết Student std = obj;

8.5 ú pháp lớp gốc

- Toán tử instanceof: Để ép kiểu đối tượng thành công, trước đó cần chắc chắn rằng đối tượng cần ép kiểu là một thể hiện của lớp kia. Do đó phải dùng toán tử instanceof để kiểm tra điều đó, ví dụ:

```
/** Giả sử myObj được khai  báo kiểu Object */
/** Thực hiện ép kiểu nếu myObj là một instance của
   Cylinder */
if (myObj instanceof Cylinder) {
  Cylinder myCyl = (Cylinder)myObj;
  System.out.println("The tich hinh tru la " +
   myCyl.findVolume();
   …
}
```

8.6 hạm vi truy cập

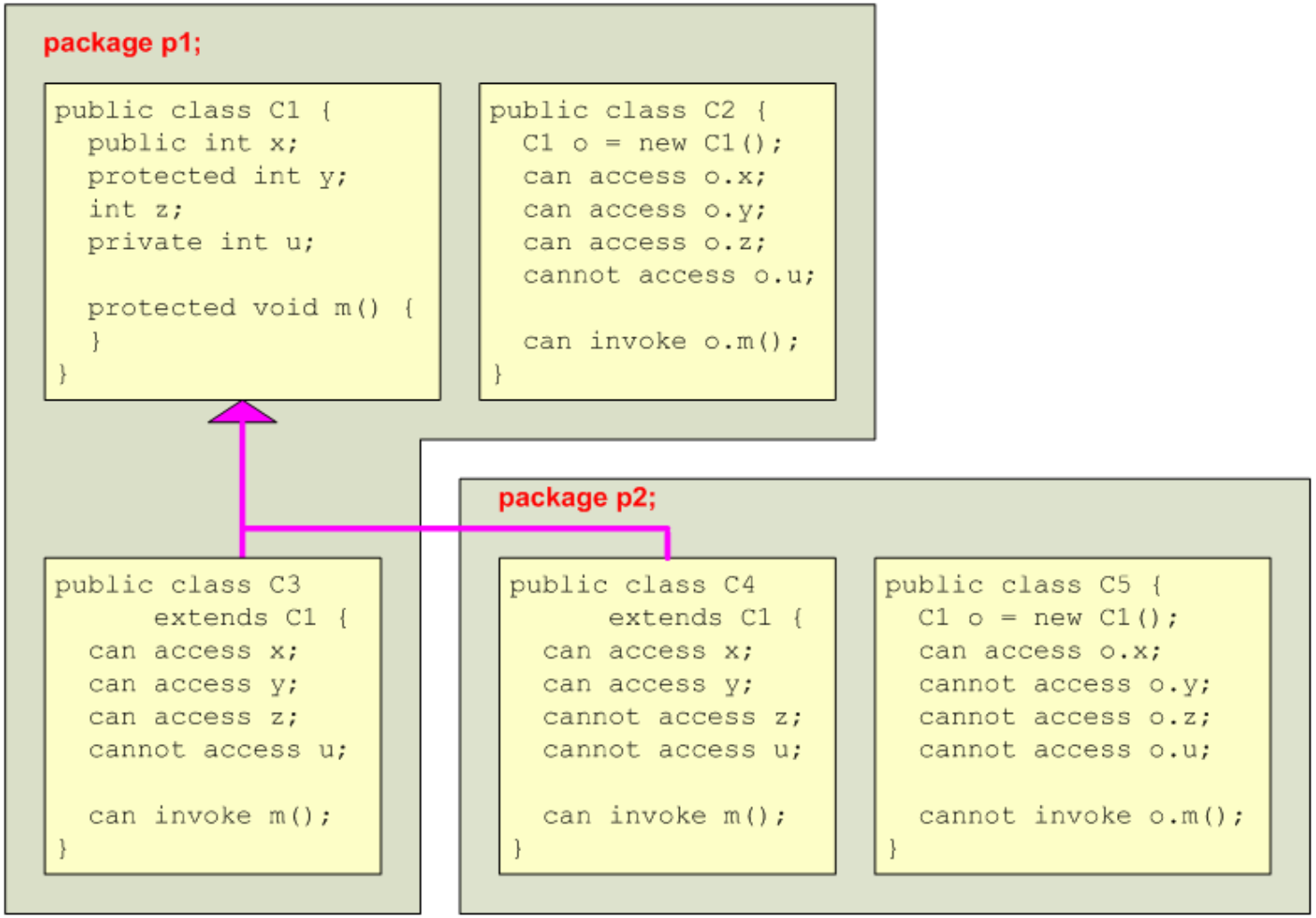
- Phạm vi của định danh: Là phần mã chương trình ở đó nó được phép sử dụng định danh đó.
- Khi một định danh cục bộ được khai báo cùng tên với một phần tử lớp, phần tử này là ẩn cho đến khi kết thúc thực hiện khối đó.
- Phần tử lớp ẩn có thể được truy cập bằng việc sử dụng từ khoá this cùng với phần tử lớp đó.
- Bốn mức truy cập phần tử lớp: public, protected, default, và private

8.6 Phạm vi truy cập

Phạm vi truy cập	public	protected	default	private
Cùng gói	Có	Có	Có	Không
Khác gói có thừa kế	Có	Có	Không	Không
Khác gói	Có	Không	Không	Không

8.6 hạm vi truy cập

- Ví dụ:



8.6 hạm vi truy cập

- Khả năng truy cập tăng dần, từ **private**, **default** (no modifier), **protected**, đến **public**
- **private:** ẩn hoàn toàn các thành phần của lớp.
- **protected:** được truy cập bởi các lớp con trong bất kỳ gói nào, hoặc các lớp trong cùng gói.
- Hai từ khóa trên chỉ có thể sử dụng cho các thành phần của lớp, không thể sử dụng cho lớp.
- *default modifier* (no modifier): được truy cập từ bất kỳ lớp nào trong cùng gói.
- **public:** được truy cập từ bất kỳ lớp nào.
- **public** và *default modifier* có thể được sử dụng cho các thành phần của lớp, cũng như sử dụng cho chính lớp.
- Chú ý: Các ký hiệu -, #, + được sử dụng để biểu diễn tương ứng các từ bổ nghĩa là **private, protected**, *và* **public.**

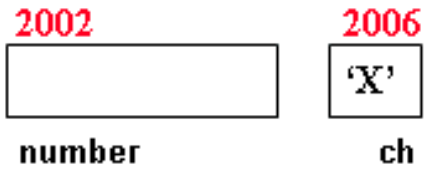
8.7 hực hiện một lớp gốc

- Một subclass constructor luôn gọi phương thức tạo của lớp cha (superclass constructor) trước tiên (hoặc ngay từ lệnh đầu tiên gọi với từ khóa super, hoặc tự động khi không có lệnh gọi đến super) để tạo và khởi tạo các thành viên thừa kế từ superclass.
- Signature của một phương thức chứa tên phương thức, số lượng và kiểu của các đối số của nó theo trật tự đã cho
- Quá tải phương thức (overloading method) là việc sử dụng tên phương thức nhiều hơn một lần, mỗi lần với một Signature khác nhau.
- Ý nghĩa của quá tải phương thức: Một số phương thức của cùng một tên được định nghĩa với các tập đối số khác nhau (trên cơ sở số lượng tham số, kiểu tham số, hoặc trật tự các tham số)

8.8 hương thức tạo sao chép

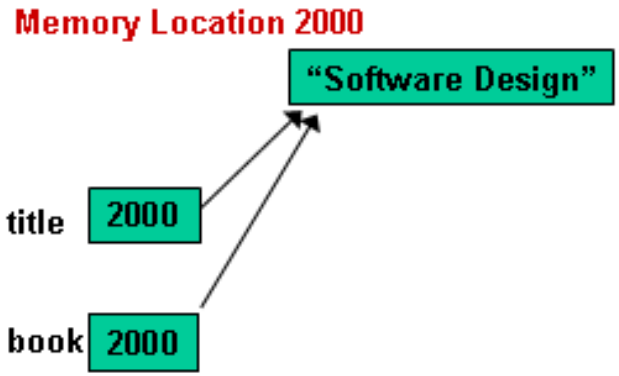
- Khi khai báo một biến thuộc kiểu dữ liệu cơ sở, vùng nhớ đủ lưu trữ một giá trị của kiểu đó được phân phối cho nó.

```
double  number;
char    ch;
ch = 'X';
```



- Khi khai báo một biến thuộc kiểu tham chiếu, nó sẽ lưu trữ địa chỉ của vùng nhớ, nơi đối tượng có thể được tìm thấy.

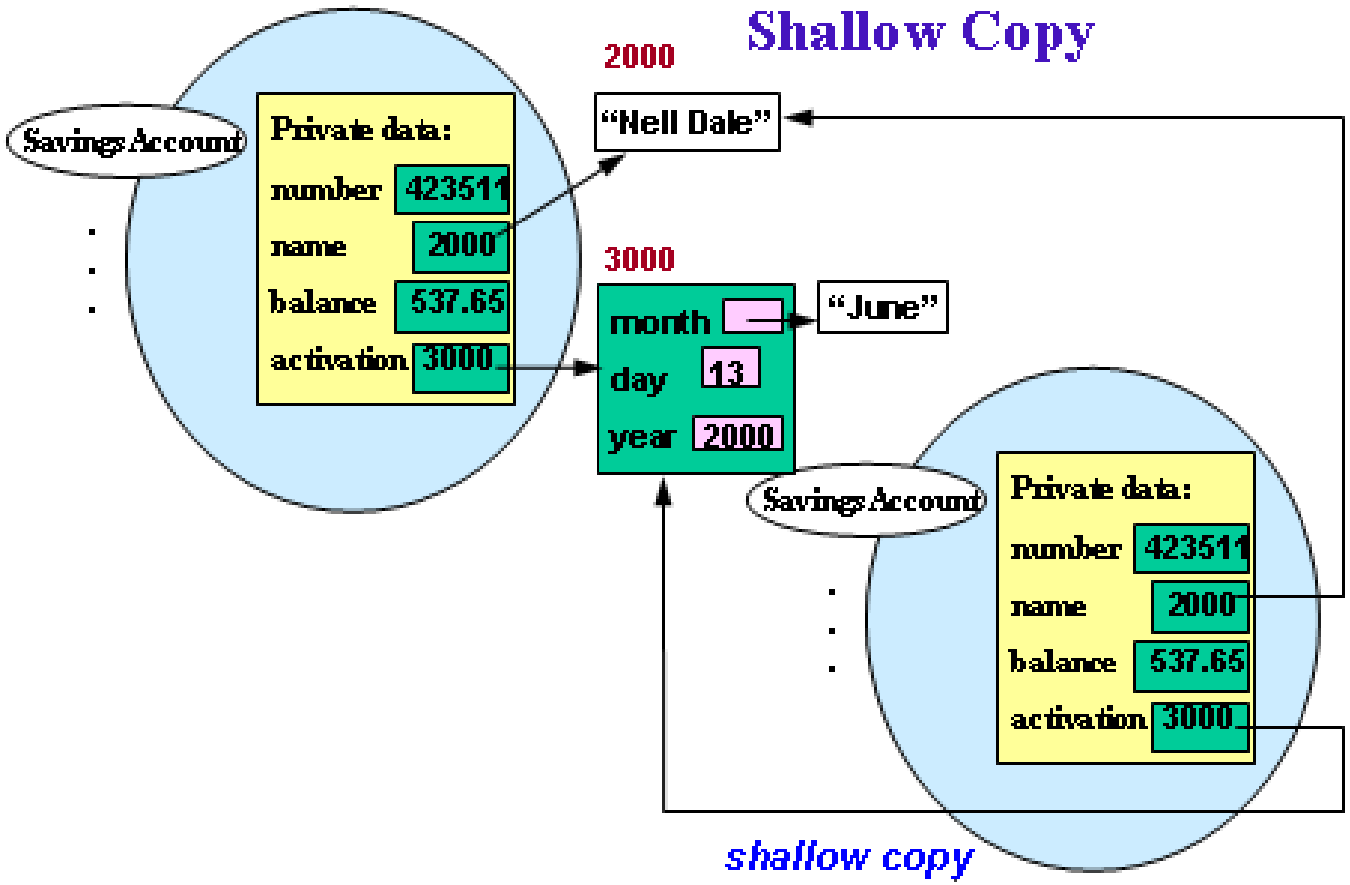
```
String  title ;
String  book ;
title = "Software Design";
book = title ;
```



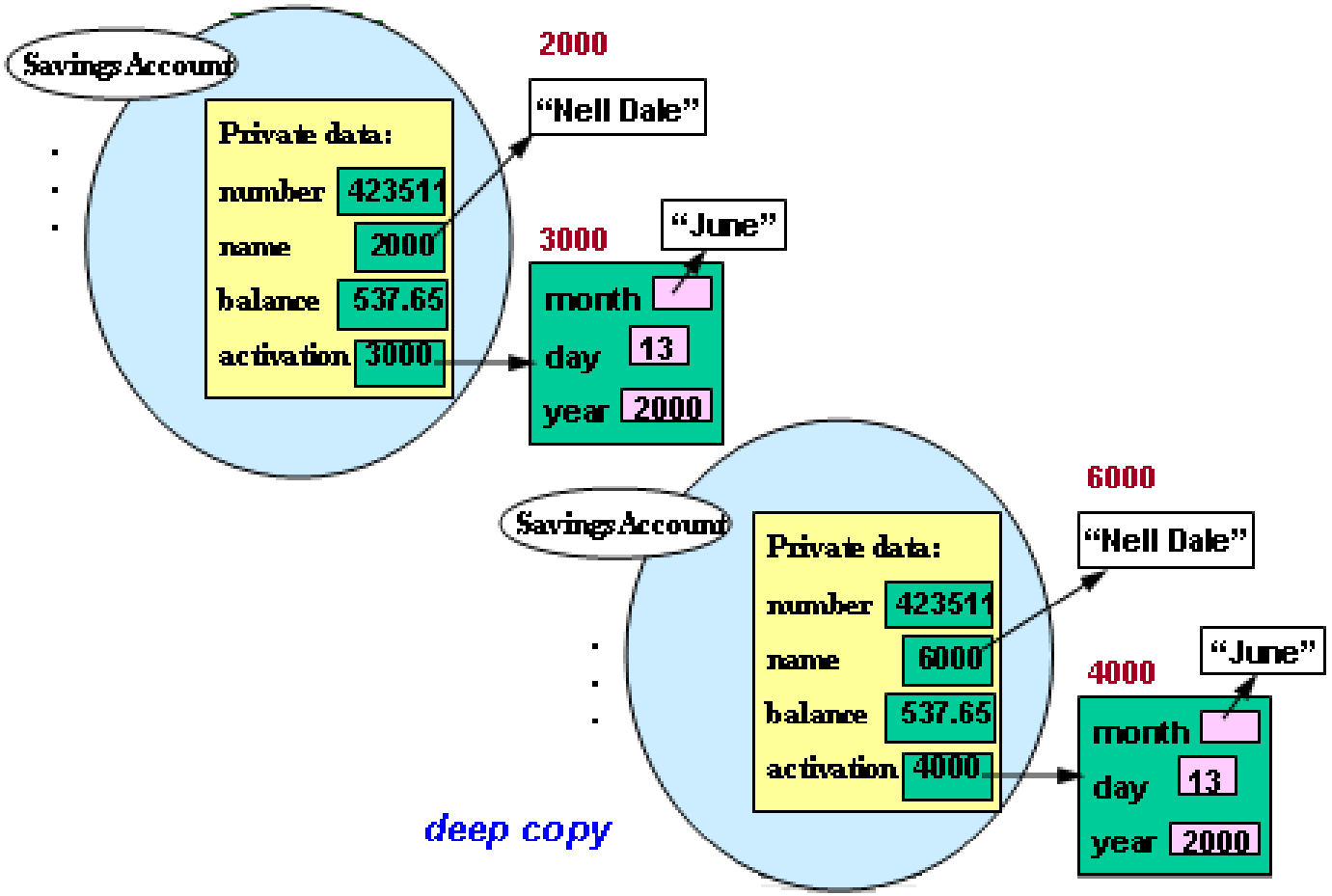
8.8 hương thức tạo sao chép

- Sao chép cạn (shallow copy) và sao chép sâu (deep copy):
- Sao chép cạn sẽ sao chép toàn bộ các trường dữ liệu lớp, bao gồm cả các tham chiếu, và không thực hiện sao chép các đối tượng tham chiếu đến bởi các trường dữ liệu.
- Sao chép sâu sẽ sao chép toàn bộ các trường dữ liệu lớp kiểu nguyên thuỷ, và còn thực hiện sao chép lưu trữ tách biệt những gì được tham chiếu đến.
- Sự khác nhau: Sao chép cạn chia sẻ các đối tượng sao chép với đối tượng lớp gốc, trong khi sao chép sâu tạo bản sao của riêng nó của các đối tượng sao chép ở những vị trí khác nhau.

8.8 hương thức tạo sao chép



8.8 hương thức tạo sao chép



8.8 hương thức tạo sao chép

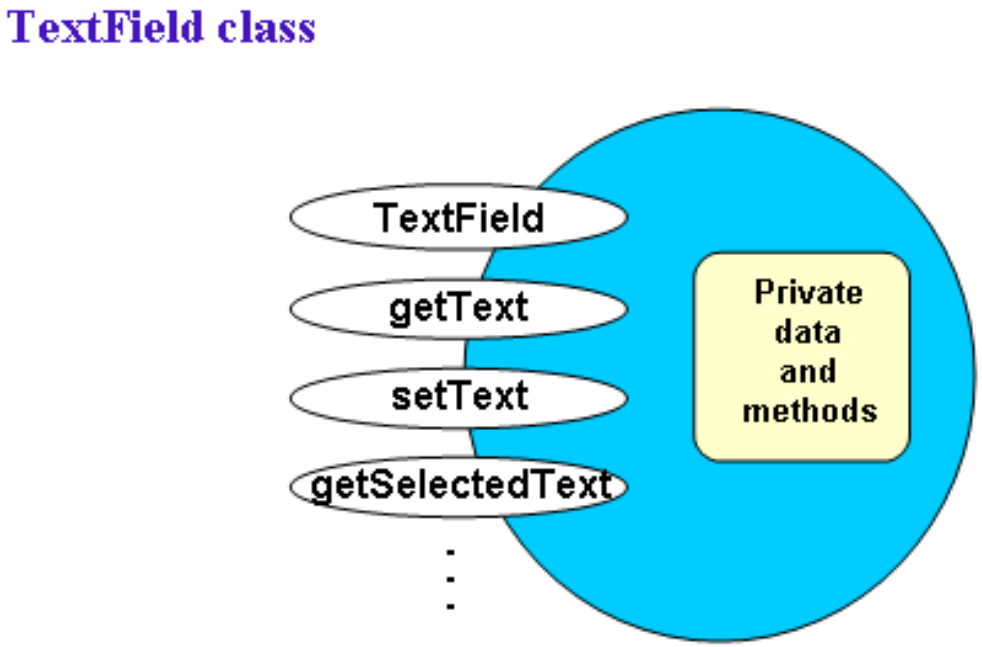
- Phương thức tạo sao chép: là phương thức tạo ra một sao chép sâu của một đối tượng có thể được sử dụng cho các mục đích khác, như tạo ra một thể hiện (instance) mới của một đối tượng không thể thay đổi từ một đối tượng cũ khác

```
public SavingsAccount(SavingsAccount oldAcct,
                      String changeOfAddress)
{
    . . .           // create deep copy of oldAcct
}
```

```
// call
account = new Savings Account (oldAcct, newAddress)
```

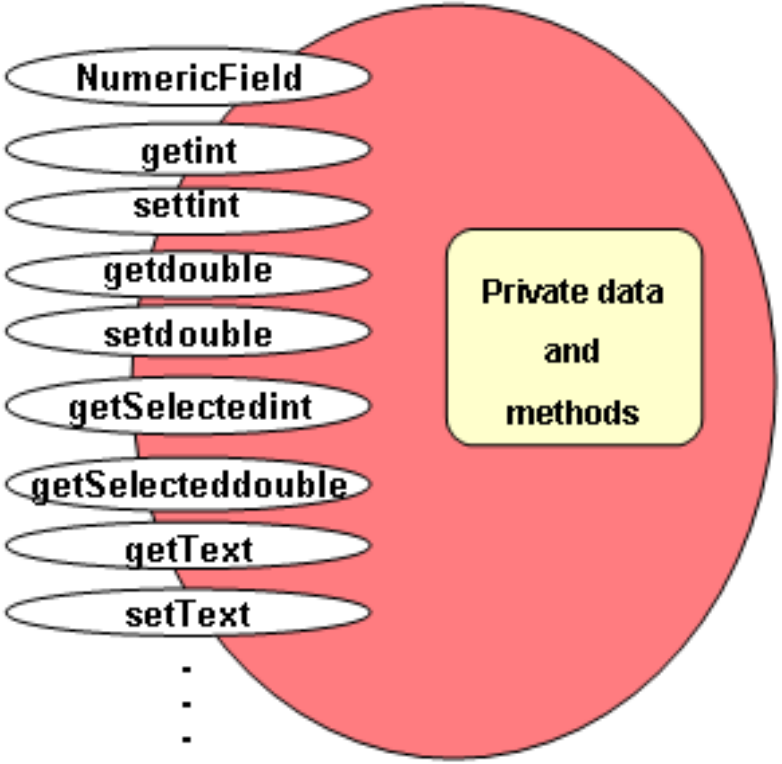
8.8 hương thức tạo sao chép

- Một số phương thức của lớp TextField



8.8 hương thức tạo sao chép

- Lớp gốc NumericField: cung cấp các phương thức để thiết lập và lấy trực tiếp các giá trị số từ một đối tượng dẫn xuất của lớp TextField. Nó không kiểm tra ngoại lệ của định dạng số, không viết chồng các phương thức của lớp TextField mà chỉ hỗ trợ kiểu dữ liệu int và double



Câu hỏi và bài tập

1. Lớp tổng quát nhất trong Java là gì?
2. Kỹ thuật cho phép một lớp mở rộng một lớp khác được gọi là gì?
3. Hãy nêu hai dạng phạm vi mà bạn biết
4. Điều gì sẽ xảy ra nếu ta không thêm một constructor vào trong lớp mới của mình?
5. Sự khác nhau giữa sao chép cạn và sao chép sâu?
6. Trong điều kiện nào thì sao chép cạn và sao chép sâu là giống nhau?

Câu hỏi và bài tập

7. Điều gì sẽ xảy ra nếu một lớp con định nghĩa một phương thức cụ thể có cùng dạng heading với một phương thức trong lớp cha của nó?
8. Điều gì sẽ xảy ra nếu một lớp con định nghĩa một trường dữ liệu có cùng tên với một trường dữ liệu trong lớp cha của nó?
9. Phân biệt giữa chồng phương thức (overriding) và ẩn trường dữ liệu (hidding).
10. Phần nào của giao diện lớp cha là không thể được thừa kế?
11. Cú pháp của một constructor khác cú pháp của một phương thức như thế nào?

Câu hỏi và bài tập

12. Xây dựng lớp Circle (hình tròn) có thuộc tính r (bán kính) mặc định là 1.0. Các phương thức tạo không tham số và phương thức tạo có tham số thiết lập giá trị cho r. Phương thức findArea() để tính diện tích của Circle.
13. Xây dựng lớp Cylinder (hình trụ) thừa kế từ Circle, có thêm thuộc tính d (chiều cao) mặc định là 1.0. Các phương thức tạo không tham số và phương thức tạo có tham số thiết lập giá trị cho d. Phương thức findArea() để tính diện tích bề mặt của Cylinder.
14. Viết chương trình tạo một đối tượng Circle và một đối tượng Cylinder. In ra diện tích của mỗi đối tượng

Chương 9

NHẬP XUẤT FILE, LẶP VÀ NGOẠI LỆ

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 9

- Nhập xuất file
- Lặp
- Toán tử điều kiện và lệnh nhảy
- Kỹ thuật quản lý ngoại lệ

9.1 hập xuất file

- Việc lưu trữ dữ liệu trong các biến, các mảng có tính chất tạm thời
- Dữ liệu sẽ mất khi biến ra khỏi tầm ảnh hưởng của nó hoặc khi chương trình kết thúc.
- File giúp chương trình có thể lưu trữ một lượng lớn dữ liệu, cũng như có thể lưu trữ dữ liệu trong một thời gian dài ngay cả khi chương trình kết thúc.

9.1 hập xuất file

- Khai báo: Để nhập xuất sử dụng file, cần các khai báo sau
  - Nạp gói java.io.*
  - Chọn các tên và kiểu phù hợp cho các biến file và khai báo chúng.
  - Tạo một đối tượng file cho mỗi biến file.
  - Sử dụng các tên file trong các lệnh nhập-xuất
  - Đóng các file khi đã xong việc.
  - Tiến trình khởi tạo file sẽ kết hợp tên file với tên vật lý của file; chèn một con trỏ file đến điểm bắt đầu của file, trỏ vào ký tự đầu tiên; nếu file không tồn tại trên đĩa, một file rỗng được tạo ra; nếu file đã tồn tại trên đĩa, nó sẽ bị xoá đi.

9.1 hập xuất file

- Khái niệm luồng: Tất cả những hoạt động nhập/xuất dữ liệu đều được quy về một khái niệm gọi là luồng (stream).
- Luồng là nơi có thể “sản xuất” và “tiêu thụ” thông tin.
- Luồng thường được hệ thống xuất nhập trong java gắn kết với một thiết bị vật lý.
- Tất cả các luồng đều có chung một nguyên tắc hoạt động.
- Cùng một lớp, phương thức nhập xuất có thể dùng chung cho các thiết bị vật lý khác nhau.
- Java định nghĩa hai kiểu luồng: luồng byte và luồng ký tự
- Luồng byte hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.
- Luồng ký tự được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode).

9.1 hập xuất file

- Luồng byte (Byte Streams): Các luồng byte được định nghĩa dùng hai lớp phân cấp.
- Mức trên cùng là hai lớp trừu tượng InputStream và OutputStream.
- InputStream định nghĩa những đặc điểm chung cho những luồng nhập byte.
- OutputStream mô tả cách xử lý của các luồng xuất byte.
- Các lớp dẫn xuất từ hai lớp InputStream và OutputStream sẽ hỗ trợ chi tiết tương ứng với việc đọc ghi dữ liệu trên những thiết bị khác nhau.

9.1 hập xuất file

Lớp luồng byte	Ý nghĩa
BufferedInputStream	Luồng nhập có đệm
BufferedOutputStream	Luồng xuất có đệm
ByteArrayInputStream	Luồng nhập dữ liệu từ một mảng byte
ByteArrayOutputStream	Luồng xuất dữ liệu đến một mảng byte
DataInputStream	Luồng nhập có những phương thức đọc những kiểu dữ liệu chuẩn trong java
DataOutputStream	Luồng xuất có những phương thức ghi những kiểu dữ liệu chuẩn trong java
FileInputStream	Luồng nhập cho phép đọc dữ liệu từ file
FileOutputStream	Luồng xuất cho phép ghi dữ liệu xuống file
FilterInputStream	Triển khai lớp trừu tượng InputStream
FilterOutputStream	Hiện thực lớp trừu tượng outputStream

9.1 hập xuất file

InputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp luồng nhập kiểu Byte
OutputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp xuất nhập kiểu Byte
PipedInputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng xuất kiểu ống
PipedOutputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng nhập kiểu ống để tạo nên một kết nối trao đổi dữ liệu kiểu ống.
PrintStream	Luồng xuất có chứa phương thức print() và println()
PushbackInputStream	Là một luồng nhập kiểu Byte hỗ trợ thao tác trả lại (push back) và phục hồi thao tác đọc một byte (unread)
RandomAccessFile	Hỗ trợ các thao tác đọc, ghi đối với file truy cập ngẫu nhiên
SequenceInputStream	Là một luồng nhập được tạo nên bằng cách nối kết logic các luồng nhập khác.

9.1 hập xuất file

- Luồng ký tự (Character Streams): Các luồng ký tự được định nghĩa dùng hai lớp phân cấp.
- Mức trên cùng là hai lớp trừu tượng Reader và Writer.
- Lớp Reader dùng cho việc nhập dữ liệu của luồng
- Lớp Writer dùng cho việc xuất dữ liệu cua luồng.
- Những lớp dẫn xuất từ Reader và Writer thao tác trên các luồng ký tự Unicode.

9.1 hập xuất file

Lớp luồng ký tự	Ý nghĩa
BufferedReader	Luồng nhập ký tự đọc dữ liệu vào một vùng đệm.
BufferedWriter	Luồng xuất ký tự ghi dữ liệu tới một vùng đệm.
CharArrayReader	Luồng nhập đọc dữ liệu từ một mảng ký tự
CharArrayWriter	Luồng xuất ghi dữ liệu tời một mảng ký tự
FileReader	Luồng nhập ký tự đọc dữ liệu từ file
FileWriter	Luồng xuất ký tự ghi dữ liệu đến file
FilterReader	Lớp đọc dữ liệu trung gian (lớp trừu tượng)
FilterWriter	Lớp xuất trung gian trừu tượng
InputStreamReader	Luồng nhập chuyển bytes thành các ký tự
LineNumberReader	Luồng nhập đếm dòng
OutputStreamWriter	Luồng xuất chuyển những ký tự thành các bytes

9.1 hập xuất file

PipedReader	Luồng đọc dữ liệu bằng cơ chế đường ống
PipedWriter	Luồng ghi dữ liệu bằng cơ chế đường ống
PrintWriter	Luồng ghi văn bản ra thiết bị xuất (chứa phương thức print() và println())
PushbackReader	Luồng nhập cho phép đọc và khôi phục lại dữ liệu
Reader	Lớp nhập dữ liệu trừu tượng
StringReader	Luồng nhập đọc dữ liệu từ chuỗi
StringWriter	Luồng xuất ghi dữ liệu ra chuỗi
Writer	Lớp ghi dữ liệu trừu tượng

9.1 hập xuất file

- Luồng được định nghĩa trước (The Predefined Streams): Gói java.lang có định nghĩa lớp System, nó có ba biến luồng được định nghĩa trước là in, out và err, các biến này là các trường được khai báo static trong lớp System.
- System.out: luồng xuất chuẩn, mặc định là màn hình console. System.out là một đối tượng kiểu PrintStream.
- System.in: luồng nhập chuẩn, mặc định là bàn phím. System.in là một đối tượng kiểu InputStream.
- System.err: luồng lỗi chuẩn, mặc định cũng là màn hình console. System.err cũng là một đối tượng kiểu PrintStream giống System.out.

9.1 hập xuất file

- Sử dụng luồng Byte: InputStream và OutputStream là hai siêu lớp của tất cả những lớp luồng xuất nhập kiểu byte.
- Những phương thức trong hai siêu lớp này tạo ra các ngoại lệ kiểu IOException.
- Chúng có thể được dùng trong các lớp con. Vì vậy tập các phương thức đó là tập tối tiểu các chức năng nhập xuất mà những luồng nhập xuất kiểu byte có thể sử dụng.

9.1 hập xuất file

Phương thức	Ý nghĩa
InputStream	
int available( )	Trả về số lượng byte có thể đọc được từ luồng nhập
void close( )	Đóng luồng nhập và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ tạo ra một ngoại lệ IOException
void mark(int nByte)	Đánh dấu ở vị trí hiện tại trong luồng nhập
boolean markSupported( )	Kiểm tra xem luồng nhập có hỗ trợ phương thức mark() và reset() không.
int read( )	Đọc byte tiếp theo từ luồng nhập
int read(byte buffer[ ])	Đọc buffer.length bytes và lưu vào vùng nhớ buffer. Kết quả trả về số bytes thật sự đọc được
int read(byte buffer[ ], int offset, int numBytes)	Đọc numBytes bytes bắt đầu từ địa chỉ offset và lưu vào trong vùng nhớ buffer. Kết quả trả về số bytes thật sự đọc được
void reset( )	Nhảy con trỏ đến vị trí được xác định bởi việc gọi hàm mark() lần sau cùng.
long skip(long numBytes)	Nhảy qua numBytes dữ liệu từ luồng nhập

9.1 hập xuất file

OutputStream	
void close( )	Đóng luồng xuất và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ tạo ra một ngoại lệ IOException
void flush( )	Ép dữ liệu từ bộ đệm phải ghi ngay xuống luồng (nếu có)
void write(int b)	Ghi byte dữ liệu chỉ định xuống luồng
void write(byte buffer[ ])	Ghi buffer.length bytes dữ liệu từ mảng chỉ định xuống luồng
void write(byte buffer[ ], int offset, int numBytes)	Ghi numBytes bytes dữ liệu từ vị trí offset của mảng chỉ định buffer xuống luồng

9.1 hập xuất file

- Đọc dữ liệu từ màn hình Console

```
import java.io.*;
class ReadBytes {
    public static void main(String args[]) throws
    IOException {
        byte data[] = new byte[100];
        System.out.print("Enter some characters.");
        System.in.read(data);
        System.out.print("You entered: ");
        for(int i=0; i < data.length; i++)
            System.out.print((char) data[i]);
    }
}
```

9.1 hập xuất file

- Đọc và ghi file dùng luồng Byte
  - Phương thức *readLine()* của lớp *BuffereReader* không sử dụng đối số, trả về một chuỗi.
  - Nó đọc một dòng vào từ file, bao gồm cả dấu hiệu kết thúc dòng (end-of-line mark), nhưng loại bỏ dấu hiệu *EOL*, rồi lưu phần còn lại của dòng tại vị trí trả về của chuỗi. Ví dụ:

```
String line ;
line = inFile.readLine( ) ;
```

  - Có thể sử dụng để nhập một giá trị số từ file, ví dụ:

```
int    numberOfDependents;
line = inFile.readLine( );
numberOfDependents = Integer.valueOf(line).intValue( );
double   taxRate ;
line = inFile.readLine( );
taxRate = Double.valueOf( line ).doubleValue( );
```

9.1 hập xuất file

- Ví dụ sử dụng files:

```
import  java.io.*;            // Các lớp file nằm ở đây
public class EditLine {
 private static BufferedReader inFile; // file nhập dữ liệu
 private static PrintWriter outFile;   // file xuất dữ liệu
public static void main( String[ ] args ) throws IOException{
   // Chuẩn bị các file để đọc và ghi dữ liệu
   inFile = new BufferedReader(new FileReader("infile.dat"));
   outFile = new PrintWriter( new FileWriter("outfile.dat"));
     . . .
   inFile.close( );
   outFile.close( );
  }
}
```

9.1 hập xuất file

- Phương thức *write()* của lớp *FileWtite* có thể nhận một giá trị kiểu *int* hoặc một đối tượng thuộc lớp *String* làm đối số.
- Một đối số kiểu *int* trước hết phải được chuyển thành ký tự, sử dụng mã Unicode, sau đó được viết vào file như một ký tự. Ví dụ:

```
FileWriter outFile ;
outFile = new FileWriter(“outfile.dat”);
outFile.write(‘Q’) ;
outFile.write(“This is written to file.”);
```

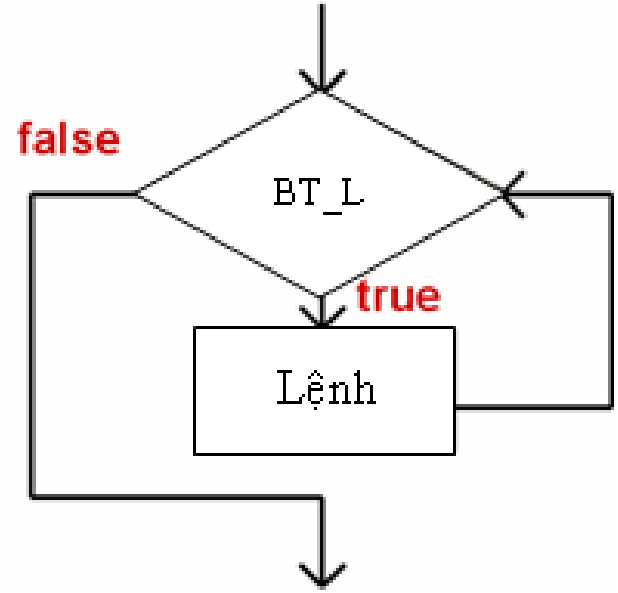
- Phương thức *print()* của lớp *PrintWrite* sử dụng một đối số thuộc kiểu dữ liệu nguyên thủy bất kỳ của Java, như *char, int, long, float, double* hoặc *String*.
- Phương thức *println()* tự động thêm dấu hiệu kết thúc dòng (*EOL*) vào cuối những gì nó ghi lên file. Ví dụ:

```
PrintWriter  outFile ;
    outFile = new PrintWriter( new FileWriter(“outfile.dat”) );
    outFile.println( “Average blood pressure is ” + avgBP
                + “ for “ + count + “ patients.” );
```

9.2 ặp

- Lệnh lặp *while*:

```
while (BT_L) {
    // thân_vòng_lặp;
    Lệnh;
}
```



9.2 ặp

- Ví dụ:

```
int i = 0;
while (i < 100) {
  System.out.println("Welcome to Java!");
  i++;
}
```

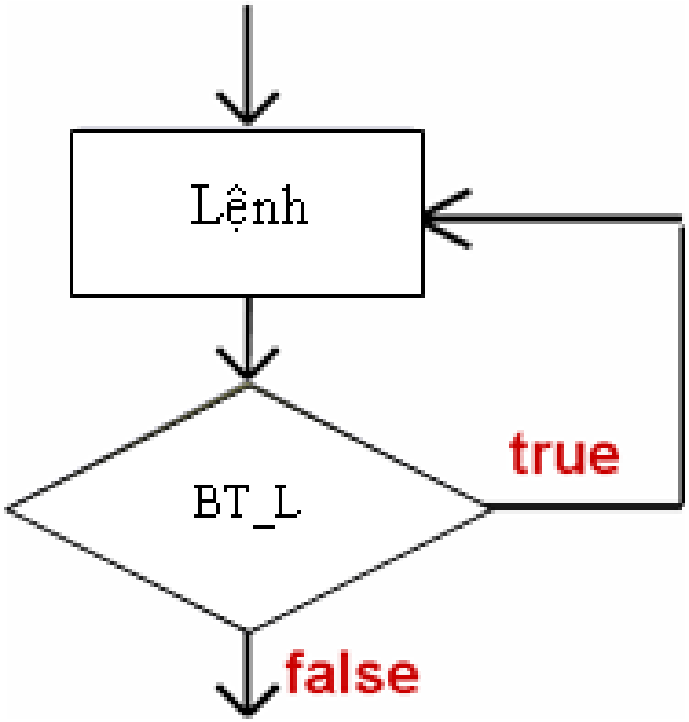
- Lưu ý: Không sử dụng giá trị dấu chấm động để kiểm tra đẳng thức trong điều khiển lặp (vì giá trị dấu chấm động là gần đúng)

```
// data should be zero
double data = Math.pow(Math.sqrt(2), 2) - 2;
if (data = = 0) System.out.println("data is zero");
else System.out.println("data is not zero");
```

9.2 ặp

- Lệnh lặp *do-while*:

```
do {
  // thân_vòng_lặp;
  Lệnh;
} while (BT_L);
```



9.2 ặp

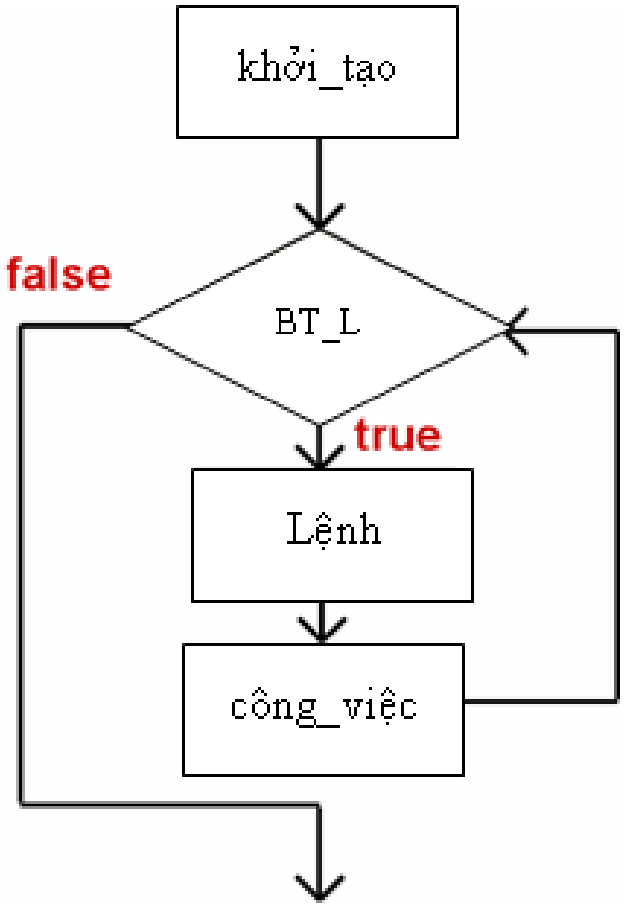
- Hoạt động: *BT_L* là biểu thức logic. Máy thực hiện các lệnh trong thân vòng lặp (Lệnh) rồi kiểm tra *BT_L*. Nếu *BT_L* nhận giá trị *true* (đúng) thì lặp lại việc thực hiện (Lệnh) và cứ tiếp tục như vậy. Ngược lại, nếu *BT_L* nhận giá trị *false* (sai) thì kết thúc vòng lặp.
- Ví dụ:

```
int i = 0;
do {
  System.out.println("Welcome to Java!");
  i++;
} while (i < 100)
```

9.2 ặp

- Lệnh lặp *for*:

```
for (khởi_tạo; BT_L; công_việc) {
   // thân vòng lặp;
   Lệnh;
}
```



9.2 ặp

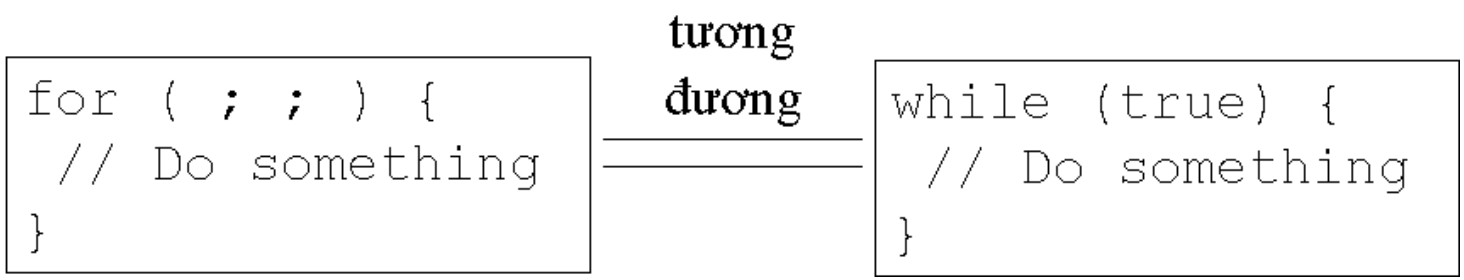
- Hoạt động: *BT_L* là một biểu thức logic. Máy thực hiện lệnh khởi_tạo (thường là khởi tạo biến đếm), rồi kiểm tra *BT_L*. Nếu *BT_L* nhận giá trị *true* (đúng) thì thực hiện các lệnh trong thân vòng lặp (Lệnh), sau đó thực hiện công việc sau mỗi lần lặp (thường là cập nhật biến đếm), rồi quay lại bước kiểm tra, và cứ tiếp tục như vậy. Ngược lại, nếu *BT_L* nhận giá trị *false* (sai) thì kết thúc vòng lặp.
- Ví dụ:

```
int i;
for (i = 0; i < 100; i++) {
  System.out.println("Welcome to Java! ” + i);
}
```

9.2 ặp

- Lưu ý: Các trường hợp sau đây là đúng (về mặt cú pháp)

```
for (int i = 1; i < 100; System.out.println(i++));
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
// Do something
}
```



9.1 Toán tử điều kiện và lệnh nhảy

- Toán tử điều kiện
  - Cú pháp:

    (BT_logic) ? bt1 : bt2 ;
  - Ý nghĩa: Nếu *BT_logic* nhận giá trị *true* (đúng) thì thực hiện *bt1*, ngược lại, thực hiện *bt2*.
  - Ví dụ:

    if (x > 0) y = 1

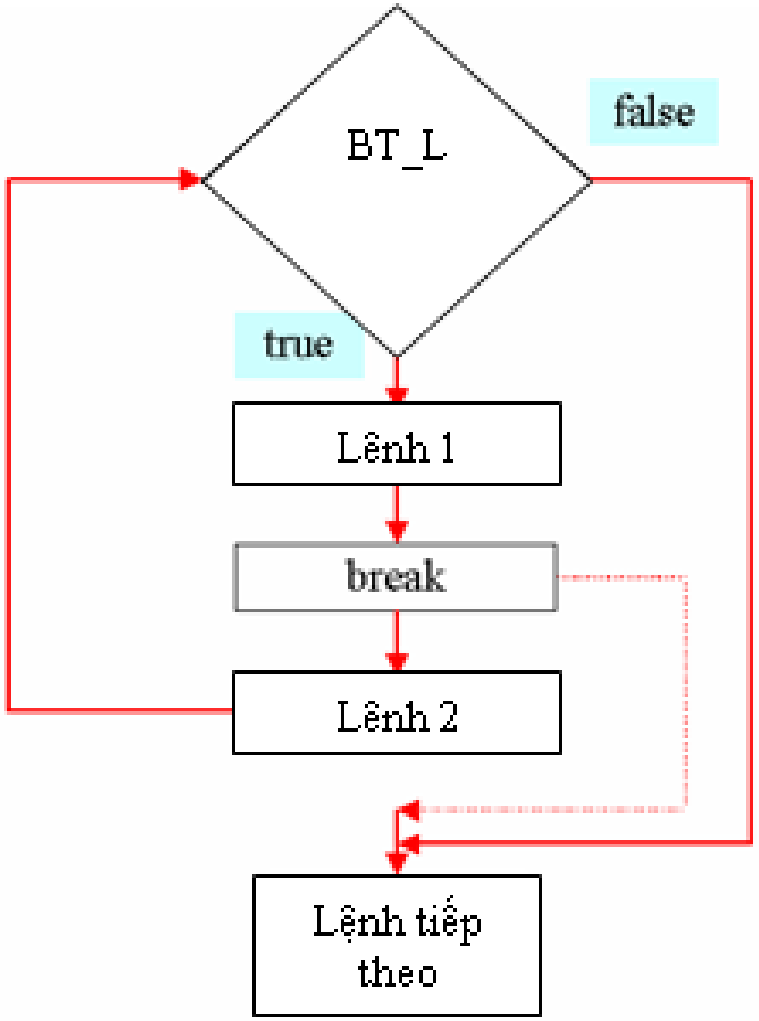
    else y = -1;

    tương đương với:

    y = (x > 0) ? 1 : -1;

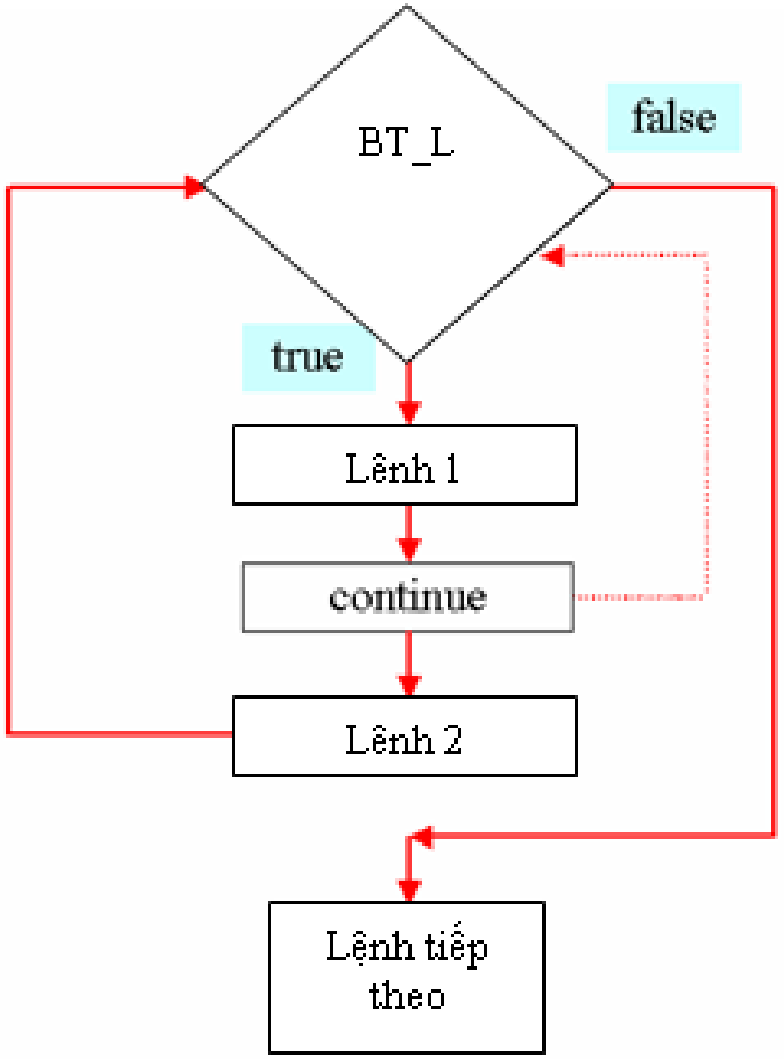
9.1 Toán tử điều kiện và lệnh nhảy

- Lệnh break: Dùng lệnh break để thoát khỏi cấu trúc switch trong cùng chứa nó.
- Tương tự, trong cấu trúc lặp, lệnh break dùng để thoát khỏi cấu trúc lặp trong cùng chứa nó.



9.1 Toán tử điều kiện và lệnh nhảy

- Lệnh continue: Dùng để tiếp tục vòng lặp trong cùng chứa nó (ngược với break)



9.1 Toán tử điều kiện và lệnh nhảy

- Nhãn (label): Không giống như C/C++, Java không hỗ trợ lệnh goto để nhảy đến một vị trí nào đó của chương trình. Java dùng kết hợp nhãn (label) với từ khóa break và continue để thay thế cho lệnh goto. Ví dụ:

```
label:
    for (…){
      for (…){
            if (<biểu thức điều kiện>)
                  break label;
            else
                  continue label;
      }
    }
```

9.1 Toán tử điều kiện và lệnh nhảy

- Định giá ngắn mạch:
  - Java sử dụng định giá ngắn mạch đối với các biểu thức logic có chứa các toán tử && và ||.
  - Các biểu thức logic với các toán tử && và || được định giá từ trái sang phải và việc định giá sẽ dừng lại ngay khi giá trị boolean đúng của toàn bộ biểu thức được tìm thấy.
  - Việc định giá biểu thức logic chỉ chứa && sẽ dừng lại và trả về giá trị *false* ngay khi xuất hiện giá trị *false* đầu tiên.
  - Việc định giá biểu thức logic chỉ chứa || sẽ dừng lại và trả về giá trị *true* ngay khi xuất hiện giá trị *true* đầu tiên.
  - Các biểu thức chứa & và | hoạt động tương tự như các biểu thức chứa && và ||, trừ một điều là chúng không có định giá ngắn mạch.

9.4 Kỹ thuật quản lý ngoại lệ

- Ngoại lệ: Trong quá trình thực hiện chương trình thường xảy ra một số sự kiện đặc biệt hoặc các lỗi như sự tràn số, truy xuất đến chỉ số mảng nằm ngoài tập chỉ số, thực hiện lệnh đọc một phần tử cuối tập tin
- Các sự kiện đó được gọi là ngoại lệ (exception).
- Java nhận biết hai dạng của ngoại lệ, đó là dạng kiểm tra được và dạng không kiểm tra được.
- Ngoại lệ không kiểm tra được có thể bỏ qua, nhưng chương trình phải nhận dạng một cách rõ ràng các ngoại lệ kiểm tra được.

9.4 Kỹ thuật quản lý ngoại lệ

- Chuyển tiếp ngoại lệ:
  - Một ngoại lệ có thể được chuyển tiếp bằng cách thêm mệnh đề *throws* vào phần heading của phương thức. Mệnh đề này sẽ chỉ ra tên của ngoại lệ được chuyển tiếp. Ví dụ:

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

  - Bằng cách này, ngoại lệ được chuyển đến phương thức gọi, cho đến khi một bộ quản lý ngoại lệ được tìm thấy, hoặc chuyển kết thúc với JVM.

9.4 Kỹ thuật quản lý ngoại lệ

- Ngoại lệ vào-ra (IOException):
  - Các lớp *FileReader*, *BufferedReader*, và *FileWriter* đều có thể phát ra một ngoại lệ gọi là *IOException*.
  - Phương thức *PrintWriter* không thể phát ra bất cứ ngoại lệ nào, nhưng phương thức tạo *PrintWriter* cần được chuyển một đối tượng *FileWriter* mà nó có thể phát sinh ngoại lệ vào-ra.
- Ba công đoạn của quá trình quản lý ngoại lệ:
  - Định nghĩa ngoại lệ: thông qua việc mở rộng kiểu ngoại lệ, và việc được cung cấp một cặp các phương thức tạo gọi đến lớp cha.
  - Phát sinh ngoại lệ: qua việc sử dụng mệnh đề throw.
  - Quản lý ngoại lệ: qua việc sử dụng một mệnh đề throws để xác định kiểu của ngoại lệ được chuyển tiếp, hoặc qua việc sử dụng try-catch-finally để nắm bắt được ngoại lệ.

9.4 Kỹ thuật quản lý ngoại lệ

- Ví dụ về chương trình tính thương 2 số thực
  - Định nghĩa một lớp ngoại lệ:

```
//Định nghĩa một lớp ngoại lệ để xử lý lỗi thiết lập dữ liệu
    class DataSetException extends Exception {
      public DataSetException( ) {
        super( );
      }
      public DataSetException( String message ) {
        super( message );
      }
    }
```

9.4 Kỹ thuật quản lý ngoại lệ

```
import  java.io.*;
public class SoNguyen{
  static void processFile(BufferedReader inFile){
        try{                                  // Nắm bắt ngoại lệ
          dataLine = inFile.readLine( );
          double a = Double.valueOf(dataLine).doubleValue( );
          dataLine = inFile.readLine( );
          double b = Double.valueOf(dataLine).doubleValue( );
          if (b == 0.0) throw new DataSetException("Zero in ");
          else System.out.println(""+(a/b));
        } catch (IOException except){
                  System.out.println("IOException with site ");
        } catch (NumberFormatException except){
          System.out.println("NumberFormatException in site ");
        }catch (DataSetException except){
                  System.out.println(except.getMessage( ));
        }
}
```

9.4 Kỹ thuật quản lý ngoại lệ

```
public static void main( String[ ] args ) throws FileNotFoundException,
    IOException {
    BufferedReader  inFile;        // file dữ liệu
    inFile = new BufferedReader(new FileReader("inData.dat"));
    String dataSetName;
    do
     {
        processFile(inFile);
        dataSetName = inFile.readLine( );
     } while (dataSetName != null);
     inFile.close( );
     }
}
```

Câu hỏi và bài tập

1. Nếu một chương trình cần nhập 1000 số nguyên, phương pháp nhập nào là thích hợp?
2. Câu lệnh nào sẽ đọc một dòng trong một file nhập infile và lưu số nguyên tương ứng vào biến number?
3. Hãy viết dòng đầu tiên của một lệnh while mà nó lặp lại cho đến khi giá trị của biến kiểu Boolean là done trở thành true.
4. Dạng lệnh lặp nào bạn sẽ sử dụng trong một chương trình để đọc giá đóng cửa của cổ phiếu đối với mỗi ngày trong tuần?
5. Năm bước trong việc sử dụng file nhập là gì?
6. Phương thức read đối với lớp FileReader sẽ trả về cái gì?

Câu hỏi và bài tập

7. Hãy viết đoạn chương trình đếm số lần xuất hiện của số nguyên 28 trong một file chứa 100 số nguyên.
8. Viết đoạn chương trình xuất ra dãy giờ và phút trong một ngày, bắt đầu lúc 1:00 sáng và kết thúc lúc 12:59 sáng.
9. Thiết kế và viết một ứng dụng Java để nhập vào một số nguyên n lớn hơn 1 và tính tổng bình phương các số từ 1 đến n. Ví dụ, nếu nhập vào số 4, tổng sẽ là 30 (bằng 1 + 4 + 9 + 16).
10. Sử dụng vòng lặp *for*, viết một phương thức nhận hai biến *int* là *x* và *n*, và trả về giá trị x^n.
11. Viết ứng dụng nhập vào hai chữ cái là tên viết tắt của một trong 64 tỉnh (thành) trong cả nước và hiển thị tên đầy đủ của tỉnh (thành) tương ứng. Nếu tên viết tắt là không hợp lệ, hãy hiển thị một thông báo lỗi và yêu cầu nhập tên viết tắt khác. Tên đầy đủ và viết tắt của 64 tỉnh (thành phố) do người lập trình tự quy định. Chương trình có hai nút bấm để nhập dữ liệu và kết thúc thực hiện

Chương 10

MẢNG

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 10

- Mảng một chiều
- Ví dụ về khai báo và xử lý mảng
- Mảng các đối tượng
- Mảng và phương thức
- Các dạng đặc biệt của xử lý mảng
- Mảng hai chiều
- Mảng nhiều chiều
- Lớp Vector

10.1 Mảng một chiều

- Mảng là một cấu trúc dữ liệu biểu diễn một tập hợp có thứ tự hữu hạn các phần tử cùng kiểu dữ liệu.
- Là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số, ví dụ a[1], b[2][5].
- Kiểu mảng là kiểu tham chiếu.
- Chỉ mục (hoặc chỉ số dưới) phải là kiểu số nguyên.
- Phần tử đầu tiên có chỉ mục là 0, phần tử thứ hai có chỉ mục là 1…
- Các phần tử được tự động gán giá trị khởi tạo là 0 (kiểu số), là *false* (kiểu boolean), và *null* đối với kiểu tham chiếu.
- Mảng một chiều chỉ có một chỉ mục

10.1 Mảng một chiều

- Khai báo mảng một chiều:

  DataType[] ArrayName;

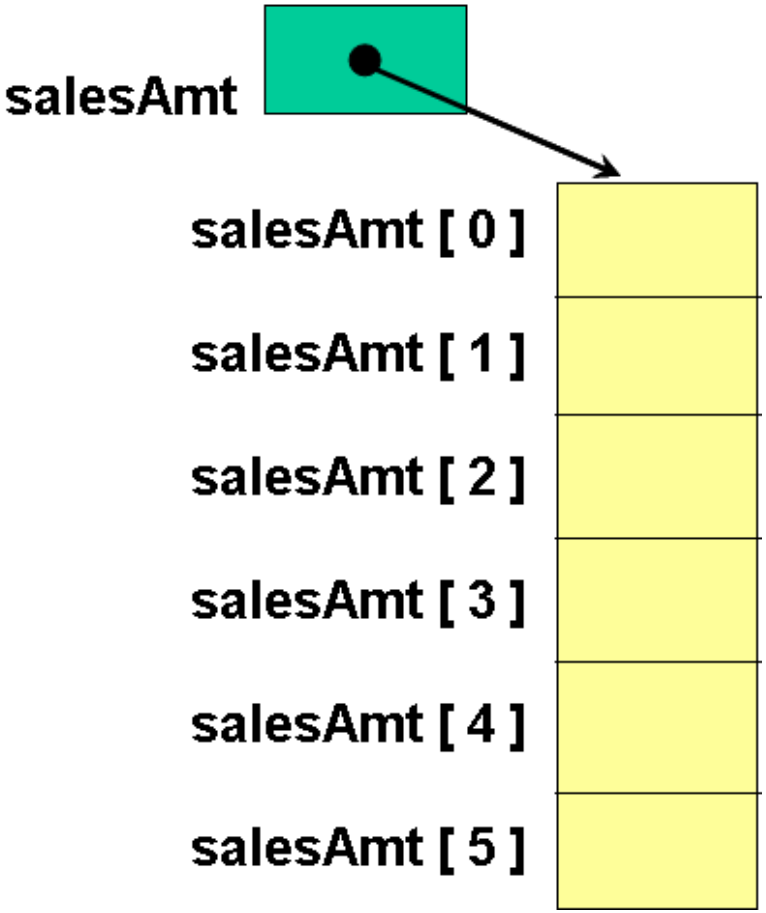
  hoặc DataType ArrayName[];

  - Trong đó DataType là kiểu dữ liệu của các phần tử mảng, ArrayName là tên biến mảng.
- Tạo mảng: ArrayName = new DataType[num];
  - Tạo mảng có num phần tử
- Khai báo và tạo mảng trong cùng một lệnh:

  DataType[] ArrayName = new DataType[num];
- Ví dụ

```
double[ ] salesAmt;              //Khai báo mảng
salesAmt = new double[6];        //Cấp phát 6 vị trí nhớ
```

10.1 Mảng một chiều



10.2 Ví dụ khai báo và xử lý mảng

- Độ dài mảng: Là số các phần tử của mảng, lấy bằng cách gọi biến thể hiện (instance variable) *length*:

  arrayVariable.length

- Ví dụ:
- salesAmt.length trả về giá trị 6
- Khởi tạo các giá trị cho mảng:
  - Sử dụng vòng lặp, ví dụ:

    ```
    for (int i = 0; i < salesAmt.length; i++)
            salesAmt [i] = i;
    ```

  - Nhập giá trị cho mảng từ bàn phím, ví dụ:

    ```
    Scanner s = new Scanner(System.in);
    for (int i = 0; i < salesAmt.length; i++)
            salesAmt [i] = s.nextDouble();
    ```

- Khai báo, tạo, khởi tạo trong một lệnh: Ví dụ

  ```
  double[] myList = {1.9, 2.9, 3.4, 3.5};
  ```

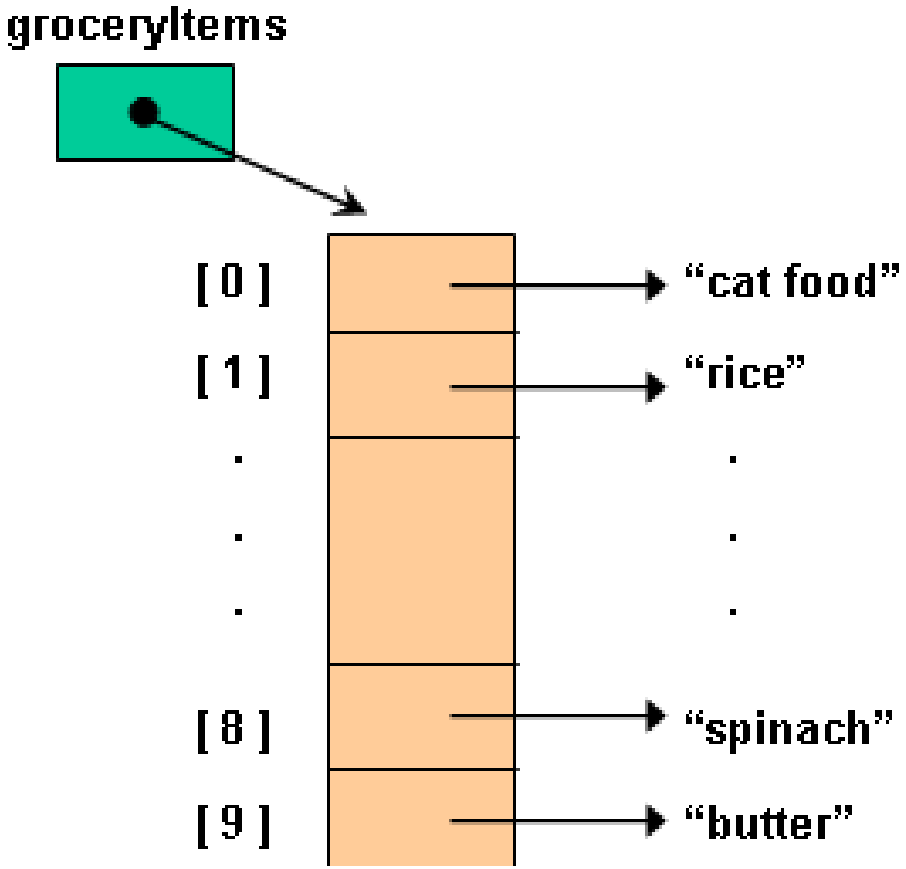
10.3 Mảng các đối tượng

- Chỉ mục của mảng: có thể là biểu thức kiểu char, sort, byte, hoặc int.
  - Không được vượt quá giới hạn: giữa khoảng 0 và độ dài mảng trừ đi 1.
- Mảng các đối tượng: Là mảng mà các phần tử của nó là các đối tượng
- Tham chiếu đến một phần tử của mảng các đối tượng là đã thực hiện hai mức tham chiếu.
  - Tham chiếu đến phần tử mảng.
  - Tham chiếu đến đối tượng được tham chiếu bởi phần tử của mảng.

10.3 Mảng các đối tượng

- Ví dụ khai báo mảng các chuỗi.

  String[ ] groceryItems = new String[10];



10.4 Mảng và phương thức

- Java sử dụng truyền tham trị (sao chép giá trị cần truyền vào các tham số) để truyền các tham số cho phương thức.
  - Tham số kiểu nguyên thủy: giá trị thực được truyền.
  - Tham số kiểu mảng: giá trị của tham số chứa một tham chiếu tới mảng; tham chiếu này được truyền
- Ví dụ: Tính giá trị trung bình của một mảng

```
public  static double  average (  int[ ]  grades ) {
        int  total = 0 ;
        for ( int i = 0 ;  i < grades.length ;  i++  )

                total = total + grades[ i ] ;
        return  (double)  total / (double) grades.length ;
}
```

10.5 Các dạng đặc biệt của xử lý mảng

- Sao chép mảng: sử dụng vòng lặp, ví dụ

```
int[ ] sourceArray = {2, 3, 1, 5, 10};
int[ ] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArrays.length; i++)
targetArray[i] = sourceArray[i];
```

- Tiện ích *arraycopy* sẽ sao chép giá trị từ mảng nguồn *sourceArray*, ở một vị trí bắt đầu *src_pos*, đến mảng đích *targetArray*, ở một vị trí bắt đầu *tar_pos*, với số phần tử cần sao chép là *length*.

```
System.arraycopy(sourceArray, src_pos, targetArray,
tar_pos, length);
```

10.5 Các dạng đặc biệt của xử lý mảng

- Tìm max, min của mảng: sử dụng vòng lặp, ví dụ

```
min = max = nums[0];
for(int i=1; i < nums.length; i++){
  if(nums[i] < min) min = nums[i];
  if(nums[i] > max) max = nums[i];
}
```

10.5 Các dạng đặc biệt của xử lý mảng

- **Sắp xếp mảng theo phương pháp nổi bọt (Bubble Sort)**

```
class BubbleSort{
    public static void main(String args[]){
      int nums[ ] = { 99, -10, 103, 18, -978, 5623, 463, -9, 287, 49 };
          int a, b, t, size = 10;
          System.out.print("Original array is:");
          for(int i=0; i < size; i++) System.out.print(" " + nums[i]);
          for(a=1; a < size; a++)
            for(b=size-1; b >= a; b--){
              if(nums[b-1] > nums[b]){ // Hoán vị phần tử
                t = nums[b-1]; nums[b-1] = nums[b];nums[b] = t;
              }
            }
          System.out.print("Sorted array is:");
          for(int i=0; i < size; i++) System.out.print(" " + nums[i]);
    }
}
```

10.6 Mảng hai chiều

- Mảng hai chiều là mảng được cấu trúc thành hai chiều
- Mỗi phần tử được truy cập đến bởi hai chỉ mục
- Khai báo mảng hai chiều:

  DataType[][] ArrayName;

  Hoặc: DataType ArrayName[][];
- Ví dụ:

  double[][] alpha;

  String[][] beta;

  int data[][];
- Tạo biến mảng hai chiều:

  ArrayName = new DataType [Expression1] [Expression2];

10.6 Mảng hai chiều

- Mảng gồ ghề: là mảng hai chiều, mỗi hàng có thể có độ dài khác nhau. Ví dụ:

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

10.6 Mảng hai chiều

- Các trường độ dài: Có hai trường độ dài, trường thứ nhất biểu diễn số hàng của mảng, trường thứ hai biểu diễn số phần tử của hàng. Ví dụ

```
int [ ] [ ] data = new int [ 6 ] [ 12 ] ;
```

- Trường thứ nhất: data.length sẽ cho giá trị là số hàng của mảng data và bằng 6
- Trường thứ hai: data[2].length sẽ cho giá trị là số cột của mảng data (số phần tử của hàng thứ 2) và bằng 12

10.7 Mảng nhiều chiều

- Định nghĩa mảng tổng quát: Mảng là một tập hợp các phần tử, tất cả đều có cùng một kiểu dữ liệu (hoặc lớp), và được cấu trúc thành *N* chiều (*N* >= 1). Mỗi phần tử của mảng được truy cập đến bởi *N* chỉ mục, mỗi chỉ mục biểu diễn vị trí của phần tử trong mỗi chiều.
- Khai báo mảng nhiều chiều:

```
final int NUM_DEPTS   =  5 ;   // mens, womens, childrens, electronics, furniture

final int NUM_STORES  =  3 ;   // White Marsh,  Owings Mills, Towson

int [ ] [ ] [ ]  monthlySales;

monthlySales = new int [ NUM_DEPTS ] [ 12 ] [ NUM_STORES ] ;
```

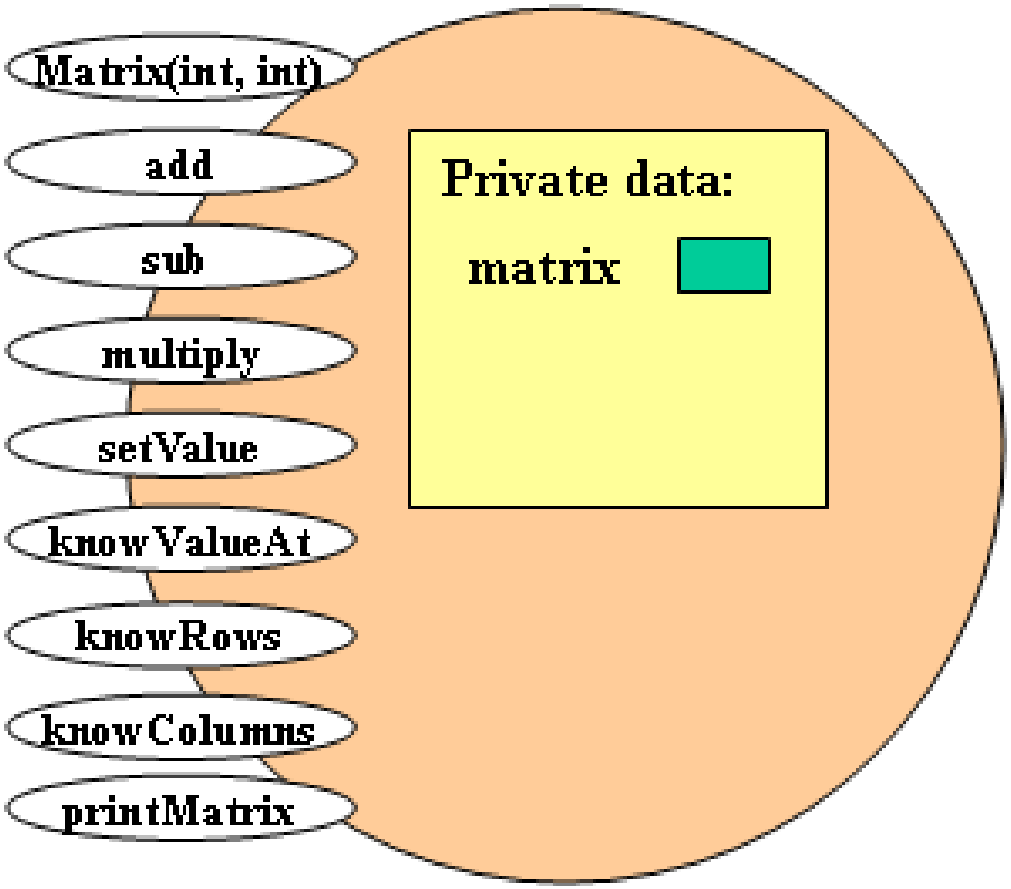
rows columns sheets

10.8 Lớp Vector

- Lớp Vector có trong gói *java.util*.
  - Chức năng tương tự như của mảng.
  - Mảng là cấu trúc triển khai cơ bản được sử dụng trong lớp này.
  - Một Vector có thể lớn lên và nhỏ đi; kích thước của nó là không cố định trong thời gian sống của nó.
- Các phép toán trên ma trận: Trong toán học, có nhiều bài toán, như các phép quay trong đồ hoạ, yêu cầu phép cộng, trừ, nhân, và chia hai ma trận. Thiết kế và triển khai lớp Matrix (lớp ma trận) tổng quát sẽ cung cấp các phép toán này cho các ma trận thực.

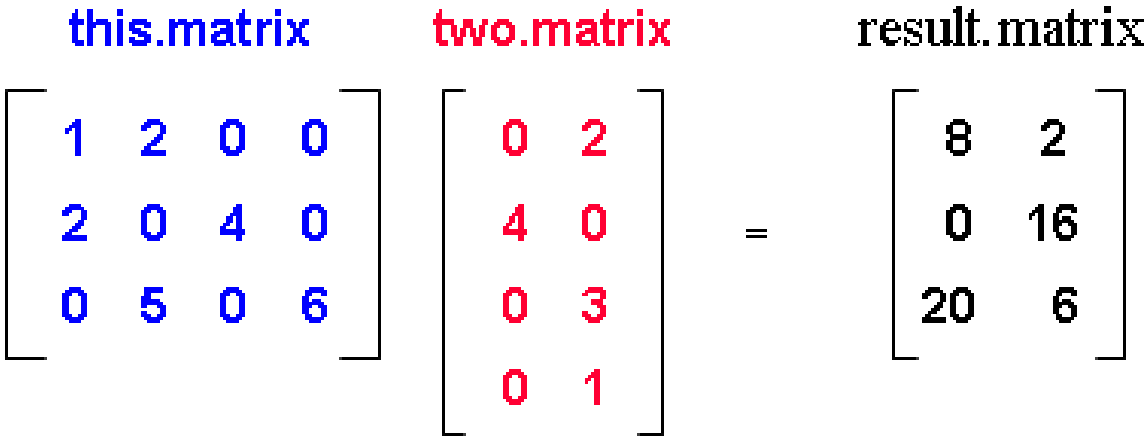
10.8 Lớp Vector

Array-based class Matrix



10.8 Lớp Vector

- Ví dụ, để tạo ma trận chứa tích của hai ma trận



this.matrix[0].length == two.matrix.length

```
Matrix  result = new Matrix ( this.matrix.length, two.matrix[0].length );
```

Câu hỏi và bài tập

1. Khai báo và tạo một mảng quizAnswer chứa 12 phần tử được đánh chỉ mục bởi các số nguyên từ 0 đến 11. Kiểu dữ liệu của các phần tử là boolean.
2. Khai báo và tạo một mảng một chiều năm phần tử kiểu *int* có tên là *oddNums* bằng cách sử dụng một danh sách khởi tạo để chứa năm số lẻ, bắt đầu từ 1.
3. Tất cả các phần tử trong một mảng cần phải có cùng một kiểu dữ liệu, và số lượng các phần tử là cố định tại thời điểm tạo mảng, đúng hay sai?
4. Các phần tử của một mảng cần phải thuộc một kiểu nguyên thuỷ, đúng hay sai?
5. Hãy viết đoạn mã tìm giá trị lớn nhất trong một mảng hai chiều double gồm 50 hàng và 50 cột.

Chương 11

DANH SÁCH VÀ ĐỆ QUY

Lê Tân

Bộ môn: Lập trình máy tính

Nội dung chương 11

- Danh sách và lớp danh sách
- Sắp xếp các phần tử của danh sách
- Danh sách đã sắp xếp
- Tìm kiếm
- Đệ quy
- Ví dụ với các biến đơn
- Giải thuật đệ quy với biến có cấu trúc

11.1 anh sách và lớp danh sách

- Danh sách là một tập đồng nhất các phần tử, liên kết giữa các phần tử là liên kết tuyến tính.
- Liên kết tuyến tính: mỗi phần tử, trừ phần tử đầu tiên, có duy nhất một phần tử đứng trước nó, và mỗi phần tử, trừ phần tử cuối cùng, có duy nhất một phần tử đứng sau nó.
- Thường sử dụng mảng một chiều để lưu trữ danh sách.
- Độ dài (length) của một danh sách là số phần tử có trong danh sách đó.
- Khoá (key) là phần tử của lớp mà giá trị của nó được sử dụng để xác định thứ tự vật lý và/hoặc logic của các phần tử trong một danh sách.

11.1 anh sách và lớp danh sách

- Thiết kế và triển khai một lớp tổng quát biểu diễn một danh sách. Các dạng tác vụ lớp cơ bản bao gồm:
  - Constructor: Tạo một đối tượng mới của lớp.
  - Transformer: Thay đổi trạng thái bên trong của một đối tượng.
  - Observer: Cho phép quan sát trạng thái của một đối tượng mà không làm thay đổi nó.
- Các dạng tác vụ lớp mở rộng:
  - Iterator: Cho phép xử lý từng thành phần của một đối tượng.
  - Copy Constructor: Tạo một đối tượng mới của lớp bằng cách sao chép một đối tượng đã có (có thể thay đổi một số hoặc tất cả các trạng thái trong quá trình xử lý).

11.1 anh sách và lớp danh sách

- Các tác vụ của lớp danh sách (class List)
- Transformer: Bao gồm hai tác vụ là *insert* (chèn một phần tử vào danh sách) và *delete* (xoá một phần tử khỏi danh sách).

```
public void insert (String item);
//Thêm một phần tử vào danh sách
//Giả thiết: phần tử chưa tồn tại trong danh sách
```

Và

```
public void delete (String item);
// Xóa phần tử (nếu có) trong danh sách
```

11.1 anh sách và lớp danh sách

- Observer: Gồm các tác vụ *isEmpty* (kiểm tra danh sách rỗng), *isFull* (kiểm tra danh sách đầy), *length* (trả về độ dài của danh sách), và *isThere* (kiểm tra một phần tử có trong danh sách).

```
public boolean isEmpty ();
// Trả về true nếu danh sách rỗng, và ngược lại
```

Và

```
public boolean isFull ();
// Trả về true nếu không còn chỗ để chèn phần tử, và ngược lại
```

Và

```
public int length ();
// Trả về số phần tử có trong danh sách
```

Và

```
public boolean isThere (String item);
// Trả về true nếu item có trong danh sách, và ngược lại
```

11.1 anh sách và lớp danh sách

- Iterator: Gồm có *resetList* (khởi tạo lại vị trí hiện tại) và *getNextItem* (lấy giá trị của phần tử hiện tại và tăng vị trí hiện tại lên 1)

```
public void resetList ();
// Khởi tạo lại vị trí hiện tại
```

Và

```
public String getNextItem ();
// Giả sử: Không có transformers được gọi từ khi iteration bắt đầu
```

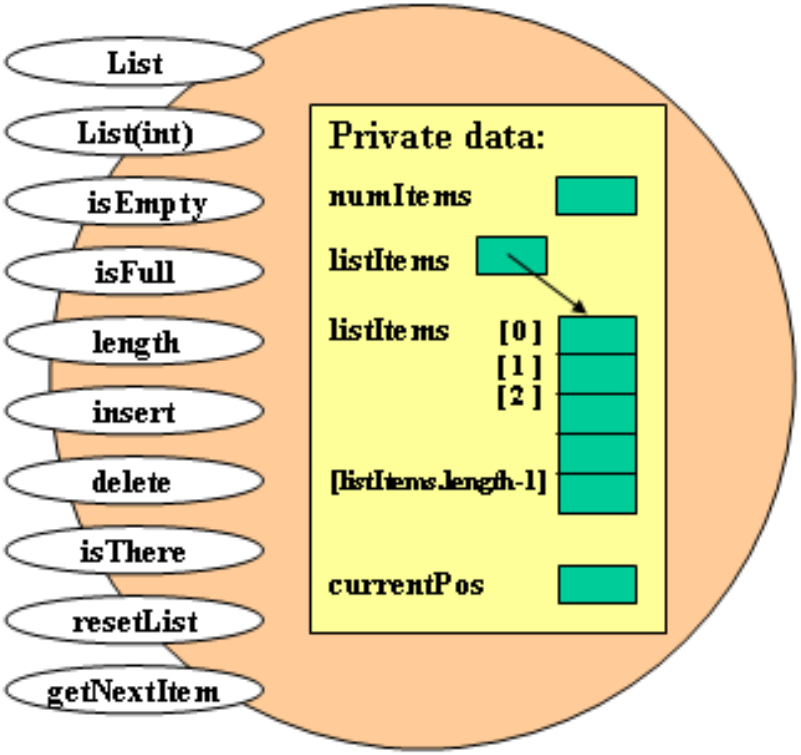
11.1 anh sách và lớp danh sách

- **Các thành phần dữ liệu của lớp danh sách:**

  numItems: Số phần tử của danh sách

  listItems[0], . . . , listItems[listItems.length - 1]: Mảng các phần tử của danh sách

  currentPos: Biến trạng thái, lưu trữ vị trí hiện tại, sử dụng cho Iterator



11.2 Sắp xếp các phần tử của danh sách

- Danh sách chưa sắp xếp và danh sánh đã sắp xếp
  - Danh sách chưa sắp xếp (unsorted list): Các phần tử được đặt vào danh sách không theo một trật tự riêng nào, theo khía cạnh nội dung của chúng.
  - Danh sách đã sắp xếp (sorted list): Các phần tử của danh sách được sắp xếp theo một trật tự nào đó, tức là được sắp xếp theo nội dung các khoá của chúng, có thể là số hoặc ký tự.

11.2 Sắp xếp các phần tử của danh sách

- Các giải thuật đáp ứng của danh sách:
  - Phương thức *insert* của danh sách:

```
public  void   insert (  String  item ) {
   if ( !this.isFull( )  )
    {
          listItems [ numItems ] = item ;
          numItems++ ;
    }
}
```

  - Phương thức *compareTo* của String: Khi so sánh dùng toán tử = =, kết quả là *true* chỉ khi nào hai tham chiếu cùng chỉ đến một đối tượng. Phương thức *compareTo* trả về 0 nếu chúng có cùng các ký tự trong cùng một trật tự, trả về một số âm nếu chuỗi là nhỏ hơn chuỗi được truyền làm tham số và trả về một số dương nếu chuỗi là lớn hơn chuỗi được truyền làm tham số.

11.2 Sắp xếp các phần tử của danh sách

```
String  s1 = new  String( "today" );
String  s2 = new  String( "yesterday" );
String  s3 = new  String( "today" );
String  s4 = new String ( "Today" );
```

s1.compareTo(s2)	-5
s1 == s3	false
s1.compareTo(s3)	0
s1.compareTo(s4)	32

11.2 Sắp xếp các phần tử của danh sách

- Phương thức *isThere* của danh sách: Trả về *true* nếu phần tử có trong danh sách, ngược lại trả về *false*.

```
public  boolean    isThere (  String  item ) {
// Trả về true nếu phần tử có trong danh sách, ngược lại trả về false.
    int     index  =  0 ;
    while  ( index < numItems
                        &&  listItems[ index ].compareTo(item) != 0 )
      index++ ;
     return  ( index < numItems ) ;
}
```

- Giải thuật *delete* của danh sách: Tìm vị trí phần tử cần xoá trong danh sách; loại bỏ phần tử cần xoá bằng cách dịch lên trước tất cả các phần tử tiếp sau đó của danh sách; giảm số phần tử của danh sách.

11.2 Sắp xếp các phần tử của danh sách

- Giải thuật sắp xếp chọn (sắp xếp tăng dần):
  - Tìm phần tử lớn nhất trong danh sách và đổi chỗ với phần tử cuối cùng của danh sách (chỉ số length – 1).
  - Tìm phần tử lớn thứ nhì trong danh sách và đổi chỗ với phần tử kế cuối của danh sách (chỉ số length – 2).
  - Lặp lại quá trình này.
  - Tìm phần tử lớn hơn trong hai phần tử còn lại của danh sách và đổi chỗ nó với phần tử ở vị trí thứ nhì của danh sách (chỉ số 1).

11.2 Sắp xếp các phần tử của danh sách

- Ví dụ:

  2, 9, 5, 4, 8, 1, 6 => 2, 6, 5, 4, 8, 1, 9 (size = 7)

  2, 6, 5, 4, 8, 1 => 2, 6, 5, 4, 1, 8 (size = 6)

  2, 6, 5, 4, 1 => 2, 1, 5, 4, 6 (size = 5)

  2, 1, 5, 4 => 2, 1, 4, 5

  2, 1, 4 => 2, 1, 4,

  2, 1 => 1, 2

  Kết quả: 1, 2, 4, 5, 6, 8, 9

11.2 Sắp xếp các phần tử của danh sách

- **Chương trình:**

```
public  void    selectSort ( ) {//  Sắp xếp tăng dần
    String temp ;
    int passCount, sIndex, maxIndex ;
      for  ( passCount = numItems - 1 ; passCount >0 ; passCount-- ){
                maxIndex = passCount ;
                for ( sIndex = passCount - 1 ; sIndex > -1 ; sIndex-- )
                          if  (listItems[ sIndex ].compareTo( listItems
                                                    [ maxIndex ]) > 0 )
                          maxIndex = sIndex ;
                temp = listItems [ maxIndex ] ;
                listItems [ maxIndex ] = listItems [ passCount ] ;
                listItems [ passCount ] = temp ;
    }
}
```

11.3 anh sách đã sắp xếp

- Giải thuật chèn một phần tử vào danh sách đã sắp xếp (sorted list) tăng dần
  - Tạo vị trí cho phần tử mới bằng cách chuyển tất cả các phần tử lớn hơn phần tử định chèn xuống dưới (về bên phải).
  - Đặt phần tử mới vào danh sách, ở vị trí vừa tạo ra
  - Tăng số phần tử của danh sách lên 1.

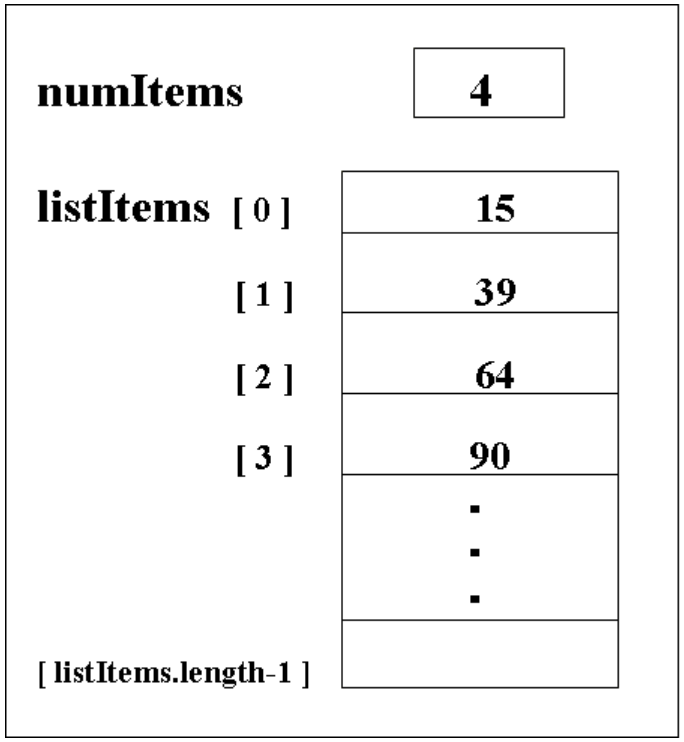
11.3 anh sách đã sắp xếp

- **Chương trình:**

```
public  void    insert ( String  item ){
    if ( !this.isFull( ) ){
      //   find proper location for new element
       int  index  =  numItems - 1 ;
      // starting at bottom of array shift down values
      // larger than item to make room for new item
       while (index >= 0 && item.compareTo(listItems[index]) < 0 ){
                listItems [ index + 1 ]  =  listItems [ index ] ;
                index-- ;
        }
        listItems [ index +1] = item ;
        numItems++ ;
    }
}
```

11.4 Tìm kiếm

- Tìm phần tử 55 trong một danh sách đã sắp xếp



Tìm kiếm tuần tự đối với 55 có thể dừng lại khi 64 đã được kiểm tra

item 55

11.4 Tìm kiếm

- Tìm kiếm tuần tự: so sánh phần tử khóa *key* với mỗi phần tử trong danh sách *list*[].
  - Việc tìm kiếm sẽ kết thúc khi tìm thấy một phần tử của danh sách bằng với *key* hoặc khi duyệt hết danh sách mà không tìm thấy phần tử nào.
  - Nếu tìm thấy, tìm kiếm tuyến tính sẽ trả về chỉ số của phần tử bằng *key*.
  - Nếu không tìm thấy, kết quả bằng -1.

11.4 Tìm kiếm

- Tìm kiếm nhị phân: Các phần tử của danh sách phải được sắp xếp theo thứ tự. Ví dụ: 2 4 7 10 11 45 50 59 60 66 69 70 79
- So sánh *key* với phần tử nằm giữa danh sách (mid).
  - Nếu *key* bằng phần tử giữa, việc tìm kiếm kết thúc vì đã tìm thấy;
  - Nếu *key* nhỏ hơn phần tử giữa, tìm *key* trong nửa đầu của mảng theo phương pháp nhị phân;
  - Nếu *key* lớn hơn phần tử giữa, tìm *key* trong nửa sau của mảng cũng theo phương pháp nhị phân.
- Lặp lại tiến trình tìm kiếm trong một nửa danh sách được xem xét tiếp theo.
- Việc tìm kiếm dừng lại khi phần tử được tìm thấy, hoặc khi không tìm thấy và cũng không còn chỗ nào để tìm nữa.

11.4 Tìm kiếm

key = 11

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
list	2	4	7	10	11	45	**50**	59	60	66	69	70	79
							↑						
key < 50							mid						

	[0]	[1]	[2]	[3]	[4]	[5]
	2	4	**7**	10	11	45
			↑			
key > 7			mid			

	[3]	[4]	[5]
	10	**11**	45
		↑	
key = 11		mid	

11.5 Đệ quy

- Định nghĩa đệ quy (recursive definition) là phương pháp định nghĩa một khái niệm trong các khái niệm của một phiên bản nhỏ hơn của chính nó. Ví dụ, định nghĩa số tự nhiên một cách đệ quy, như sau:
  - Số 1 là số tự nhiên.
  - Nếu n – 1 là số tự nhiên, thì n cũng là số tự nhiên.
- Đệ quy xuất hiện khi một phương thức gọi lại chính nó.
- Ý tưởng là đối với mỗi lời gọi đệ quy thành công sẽ tiến một bước gần với tình huống mà ở đó bài toán có thể được giải một cách dễ dàng hơn.
- Mỗi giải thuật đệ quy cần phải có ít nhất một trường hợp cơ sở, và một trường hợp đệ quy (trường hợp tổng quát).
  - Trường hợp cơ sở là trường hợp mà ở đó lời giải được chỉ ra một cách không đệ quy.
  - Trường hợp tổng quát (general case) là trường hợp mà ở đó lời giải được biểu diễn trong các số hạng của một phiên bản nhỏ hơn của chính nó.

11.5 ệ quy

- Dạng tổng quát đối với phương pháp đệ quy:

```
if (điều_kiện_cơ_sở) //Trường hợp cơ sở
   lời_giải_cụ_thể
else                          //Trường hợp tổng quát
   lời_gọi_đệ_quy
```

11.6 Ví dụ với các biến đơn

- *Ví dụ 1:* Viết một phương thức đệ quy *summation*(*n*) để tính tổng của các số nguyên từ 1 đến *n*
- *Ví dụ 2:* Viết phương thức đệ quy *factorial*(*n*) để tính *n* giai thừa
- Hàm mũ: Ta biết rằng *x*0 = 1, và *xn* = *x* * *xn-1*, với số nguyên *n* > 0. Do đó, *xn* được định nghĩa đệ quy, qua *xn*-1

11.6 Ví dụ với các biến đơn

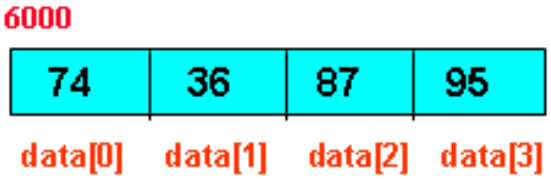
- Trường hợp cơ sở không làm gì cả đệ quy đuôi (tail recursive). Ví dụ:

```
public  static  void  printStars (  int   n ) {
//   Giả sử:  n lớn hơn hoặc bằng 0
//   Kết quả: n ngôi sao được xuất ra trên n dòng
if  ( n <= 0 )              // Trường hợp cơ sở
          else              // Trường hợp tổng quát
     {  outFile.println( "*" ) ;
        printStars ( n - 1 ) ;
   }
}
```

11.7 iải thuật đệ quy với biến có cấu trúc

- Ví dụ, phương thức đệ quy để xuất ra các phần tử của một mảng theo thứ tự ngược lại.

```
public  static void  printRev ( int[ ] data,  int  first,  int  last )
```



- Lời gọi *printRev*(*data*, 0, 3) sẽ tạo output 95 87 36 74
- Trường hợp cơ sở: số phần tử là 0, không làm gì cả.
- Trường hợp tổng quát: đưa đến gần với trường hợp cơ sở hơn, tức là độ dài của mảng giảm đi 1.
- Bằng việc xuất ra một phần tử của mảng, và xử lý mảng nhỏ hơn, ta sẽ đạt đến trường hợp cơ sở.

11.7 iải thuật đệ quy với biến có cấu trúc

- Chương trình như sau:

```
public  static  void  printRev (  int [ ] data, first , last  )
     {
     if  ( first  <=  last ) // General case
    {
      outFile.print(  data [ last ] +  “    “ ); // Xuất phần tử cuối
      printRev ( data,  first, last - 1 ) ;    // Xử lý phần còn lại
    }
                  // Trường hợp cơ sở là mệnh đề else rỗng
}
```

11.7 iải thuật đệ quy với biến có cấu trúc

- Khi viết các phương thức đệ quy, chúng ta cần chú ý một số vấn đề sau:
  - Cần có ít nhất một trường hợp cơ sở, và ít nhất một trường hợp tổng quát. Trường hợp tổng quát phải đưa ta đến gần trường hợp cơ sở hơn.
  - Các tham biến trong lời gọi đệ quy không thể giống hoàn toàn với các tham biến hình thức trong phần heading. Nếu giống, sẽ xuất hiện đệ quy vô hạn.

Câu hỏi và bài tập

1. Sự khác nhau giữa một danh sách (list) và một mảng (array) là gì?
2. Nếu một danh sách là chưa được sắp xếp, sẽ có sự cố khi ta chèn một phần tử mới vào danh sách này?
3. Hãy trình bày nguyên lý cơ bản của giải thuật tìm kiếm nhị phân.
4. Sử dụng tìm kiếm nhị phân luôn tốt hơn tìm kiếm tuyến tính, đúng hay sai?
5. Cấu trúc điều khiển xuất hiện thường xuyên nhất trong trong các phương thức đệ quy là gì?
6. Hãy viết một phương thức trả về giá trị thực hiện công thức đệ quy f(n) = f(n – 1) + f(n – 2) với trường hợp cơ sở là f(0) = 1 và f(1) = 1.
7. Hãy viết một chương trình đặt tám quân hậu trên một bàn cờ vua sao cho không có quân hậu nào ăn được bất kỳ một quân hậu khác.