



Bài 1: Tổng quan về lập trình hướng đối tượng

1

Mục tiêu bài học

- ❖ Tìm hiểu tổng quan về Công nghệ lập trình hướng đối tượng
 - Khái niệm, ưu điểm, vai trò, ứng dụng, sự phát triển các ngôn ngữ lập trình
- ❖ Tìm hiểu về Đối tượng và Lớp
 - Đối tượng, trạng thái, hành vi
 - Lớp, thuộc tính, phương thức
 - Tương tác giữa các đối tượng (thông điệp, giao diện)
- ❖ Tìm hiểu Ngôn ngữ lập trình Java
 - Giới thiệu về Java, Nền tảng Java, Mô hình dịch Java
 - Tính năng và ứng dụng của Java
- ❖ Cài đặt môi trường lập trình



2

Nội dung

1. Công nghệ hướng đối tượng (HĐT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HĐT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình



3

Nội dung

1. Công nghệ hướng đối tượng (HĐT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HĐT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình



4

3

1

1.1. Kỹ thuật lập trình

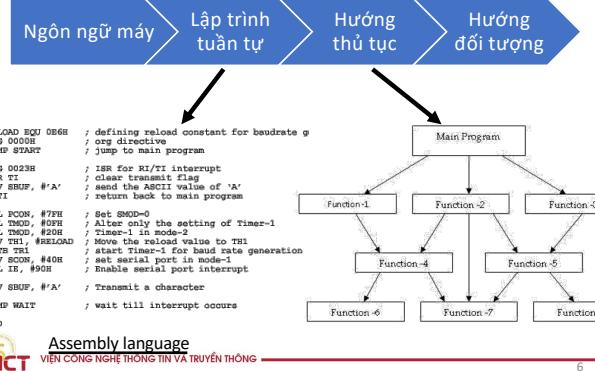
- ❖ Kỹ thuật lập trình: Kỹ thuật thực thi một giải pháp phần mềm (cấu trúc dữ liệu + giải thuật) dựa trên nền tảng một phương pháp luận (methodology) và một hoặc nhiều ngôn ngữ lập trình phù hợp với yêu cầu đặc thù của ứng dụng
- ❖ Ngôn ngữ lập trình
 - Là ngôn ngữ được chuẩn hóa
 - Cả con người và máy tính có thể đọc và hiểu được
 - Sử dụng chương trình dịch tương ứng để biên dịch toàn bộ chương trình nguồn thành mã máy trước khi thực hiện



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

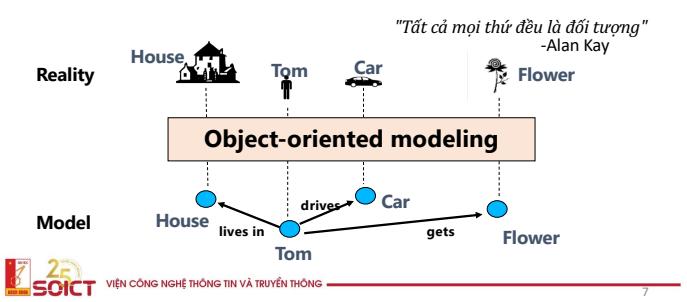
1.2. Sự phát triển của ngôn ngữ lập trình



6

1.3. Lập trình hướng đối tượng

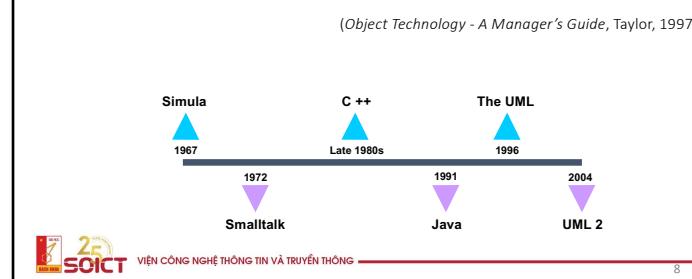
- ❖ Thể hiện các thành phần của bài toán là các “đối tượng” (object).
- ❖ Hướng đối tượng là một kỹ thuật để mô hình hóa hệ thống thành nhiều đối tượng tương tác với nhau



7

1.4 Công nghệ đối tượng (OOT)

- ❖ Công nghệ đối tượng là một tập các quy tắc (trừu tượng hóa, đóng gói, đa hình), các hướng dẫn để xây dựng phần mềm, cùng với ngôn ngữ, cơ sở dữ liệu và các công cụ khác hỗ trợ các quy tắc này.
- ❖ Các mốc chính của công nghệ đối tượng



8

OOT được sử dụng ở đâu?

- ❖ Các hệ thống Client/Server và phát triển Web
- ❖ Hệ nhúng (embedded system)
- ❖ Hệ thống thời gian thực (real-time)
- ❖ Hệ thống phần mềm nói chung...



Đối tượng là gì?

- ❖ Đối tượng trong thế giới thực, là một thực thể cụ thể mà thông thường chúng ta có thể *sờ, nhìn thấy* hay *cảm nhận* được.
- ❖ Tất cả có trạng thái (state) và hành vi (behaviour)

	Trạng thái	Hành động	
Con chó	Tên	Sửa	
	Màu	Vẫy tay	
	Giống	Chạy	
	Vui sướng	Ăn	
Xe đạp	Bánh răng	Tăng tốc	
	Bàn đạp	Giảm tốc	
	Dây xích	Chuyển bánh răng	
	Bánh xe	...	

Nội dung

1. Công nghệ hướng đối tượng (HDT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HDT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình

Trạng thái và hành vi

- ❖ Trạng thái của một đối tượng là một trong các điều kiện tại đó mà đối tượng tồn tại
- ❖ Hành vi quyết định đối tượng đó hành động và đáp trả như thế nào đối với bên ngoài
- ❖ Trạng thái của một đối tượng có thể thay đổi theo thời gian
- ❖ Hành vi nhìn thấy được của một đối tượng được mô hình thành một tập các thông điệp nó có thể đáp trả (các thao tác mà đối tượng đó thực hiện)



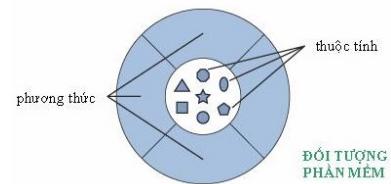
Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes



Professor Clark's behavior
Submit Final Grades
Accept Course Offering
Take Sabbatical

Đối tượng phần mềm

- Các **đối tượng phần mềm** được dùng để *biểu diễn* các đối tượng thế giới thực.
- Cũng có trạng thái và hành vi
 - Trạng thái: **thuộc tính** (attribute; property)
 - Hành vi: **phương thức** (method)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Lớp và đối tượng

- Một **lớp** là một thiết kế (blueprint) hay mẫu (prototype) cho các đối tượng cùng kiểu
 - Ví dụ: lớp XeDap là một thiết kế chung cho nhiều đối tượng xe đạp được tạo ra
- Lớp định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó
- Một **đối tượng** là một thể hiện cụ thể của một lớp.
 - Ví dụ: mỗi đối tượng xe đạp là một thể hiện của lớp XeDap
- Mỗi thể hiện có thể có những thuộc tính thể hiện khác nhau
 - Ví dụ: một xe đạp có thể đang ở số (gear) thứ 5 trong khi một xe khác có thể là đang ở số (gear) thứ 3.

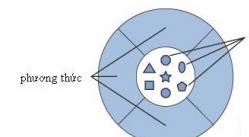


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

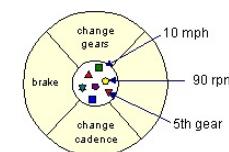
15

15

Ví dụ đối tượng phần mềm



Đối tượng phần mềm



Đối tượng phần mềm Xe Đạp

Đối tượng (object) là một thực thể phần mềm bao bọc các **thuộc tính** và các **phương thức** liên quan.

Thuộc tính được xác định bởi giá trị cụ thể gọi là **thuộc tính thể hiện**. Một đối tượng cụ thể được gọi là một **thể hiện**.



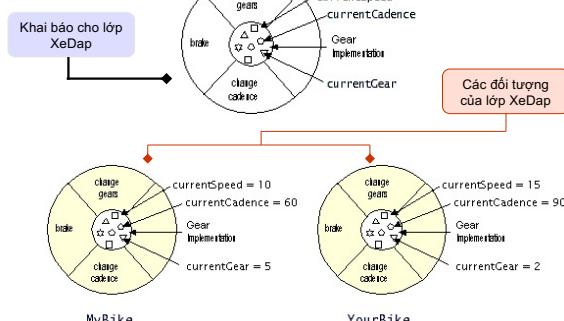
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Ví dụ Lớp Bike

Khai báo cho lớp XeDap



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

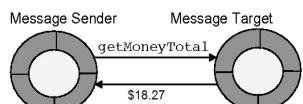
Tương tác giữa các đối tượng

- ❖ Sự giao tiếp giữa các đối tượng trong thế giới thực:



- ❖ Các đối tượng và sự tương tác giữa chúng trong lập trình

- Các đối tượng giao tiếp với nhau bằng cách gửi thông điệp (message)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

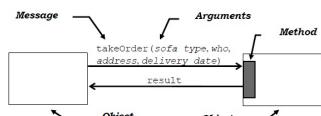
Thông điệp vs. Phương thức

❖ Thông điệp

- Được gửi từ đối tượng này đến đối tượng kia, không bao gồm đoạn mã thực sự sẽ được thực thi.

❖ Phương thức

- Thủ tục/hàm trong ngôn ngữ lập trình cấu trúc.
- Là sự thực thi dịch vụ được yêu cầu bởi thông điệp.
- Là đoạn mã sẽ được thực thi để đáp ứng thông điệp được gửi đến cho đối tượng.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

Gọi hàm vs. Gửi thông điệp

❖ Gọi hàm (Call function)

- Chỉ ra chính xác đoạn mã nào sẽ được thực hiện.
- Chỉ có duy nhất một sự thực thi của một hàm với một tên nào đó.
- Không có hai hàm trùng tên.

❖ Gửi thông điệp

- Yêu cầu một dịch vụ từ một đối tượng và đối tượng sẽ quyết định cần phải làm gì.
- Các đối tượng khác nhau sẽ có các cách thực thi các thông điệp theo cách khác nhau.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

Đặc tính của LT HĐT

Alan Kay đã tổng hợp các đặc tính của LT HĐT:

1. Tất cả đều là **đối tượng**
2. Chương trình phần mềm có thể coi là một tập hợp các đối tượng **tương tác** với nhau
3. Mỗi đối tượng trong chương trình có các **dữ liệu độc lập** của mình và **chiếm bộ nhớ riêng** của mình.
4. Mỗi đối tượng đều có dạng **đặc trưng** của lớp các đối tượng đó
5. Tất cả các đối tượng thuộc về cùng một lớp đều có các **hành vi** giống nhau



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

Hướng cấu trúc vs. Hướng ĐT

❖ Hướng cấu trúc:

- data structures + algorithms = Program
- (cấu trúc dữ liệu + giải thuật = Chương trình)

❖ Hướng đối tượng:

- objects + messages = Program
- (đối tượng + thông điệp = Chương trình)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

Nội dung

1. Công nghệ hướng đối tượng (HDT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HDT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

3.1 Trừu tượng hóa (Abstraction)

- ❖ Là quá trình loại bỏ đi các thông tin/tính chất cụ thể và giữ lại những thông tin/tính chất chung.
- ❖ Tập trung vào các đặc điểm cơ bản của thực thể, các đặc điểm phân biệt nó với các loại thực thể khác.
- ❖ Phụ thuộc vào góc nhìn (Quan trọng trong ngữ cảnh này nhưng lại không có ý nghĩa nhiều trong ngữ cảnh khác)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

Ví dụ: Trừu tượng hóa

Phụ thuộc vào góc nhìn



Registration

Owner

Garage



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

3.2 Đóng gói (Encapsulation)

- ❖ Che giấu, ẩn đi chi tiết thực hiện bên trong
- ❖ Cung cấp cho thế giới bên ngoài một giao diện
- Việc sử dụng không ảnh hưởng bởi chi tiết bên trong.

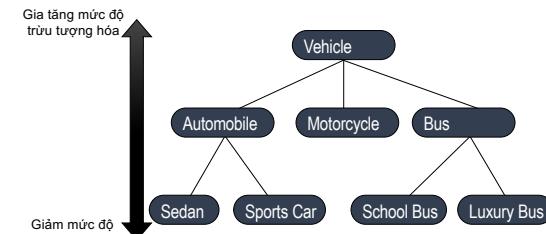


25
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

3.3. Thừa kế (Inheritance)

- ❖ Xếp hạng hay xếp thứ tự các mức trừu tượng vào một cấu trúc cây
- ❖ Các phần tử ở cùng cấp trong sơ đồ phân cấp thì có cùng mức trừu tượng hóa



26
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

3.4. Đa hình (polymorphism)

- ❖ Đa hình: “one name, many forms”
- ❖ Nạp chồng phương thức: phương thức cùng tên, nhưng hoạt động khác nhau
 - Add(int x, int y)
 - Add(float x, float y)
 - Add(float x, float y, float z)
- ❖ Ghi đè phương thức (Method Overriding)
 - Một Intern (thực tập sinh) là một Intern, đồng thời cũng có thể được xem là một Staff (nhân viên)
 - Phương thức quét thẻ của Intern khác với phương thức quét thẻ của Staff

27
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

Nội dung

1. Công nghệ hướng đối tượng (HDT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HDT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình

28
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

4.1. Java là gì?



❖ Java là một ngôn ngữ lập trình HDT được phát triển bởi Sun Microsystems, nay thuộc sở hữu của Oracle.

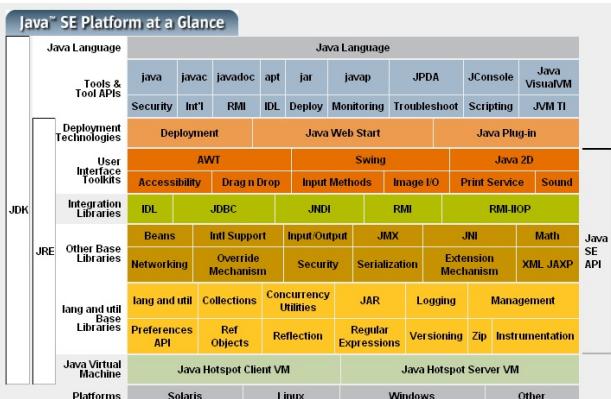
- Ban đầu được sử dụng để xây dựng ứng dụng điều khiển các bộ xử lý bên trong các thiết bị điện tử dân dụng như máy điện thoại cầm tay, lò vi sóng, ...

❖ Java - 1 ngôn ngữ lập trình - 1 công nghệ - 1 nền tảng phát triển



Green Team and James Gosling
(the leader)

Các thành phần trong Java SE



4.2 Nền tảng Java

❖ Mỗi sản phẩm Java gồm:

- Bộ công cụ phát triển: J2SDK, JDK hay SDK (development kit)
 - JDK = JRE + tools tiện ích (*.exe) + tài liệu + thư viện
- Môi trường chạy JRE (runtime environment): môi trường thực thi
 - JRE = JVM + Java runtime library (trên desktop)
 - Server JRE: JRE trên server
- Máy ảo java JVM (virtual machine)
- Các đặc tả chi tiết kỹ thuật, Ngôn ngữ lập trình, Các công nghệ đi kèm

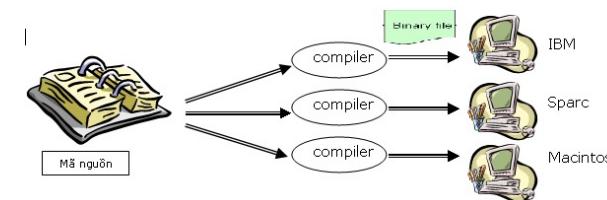
❖ Java hỗ trợ các platform

- Sun Solaris, Linux, Mac OS, FreeBSD & Microsoft Windows.

3.3. Mô hình dịch của Java

a. Mô hình biên dịch truyền thống:

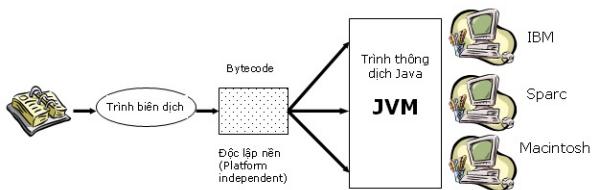
- Mã nguồn được biên dịch thành mã máy (mã nhị phân)
- Tập lệnh mã máy trên các platform khác nhau là khác nhau



3.3. Mô hình dịch của Java (2)

❖ b. Mô hình dịch của Java:

- Tiêu chí "Viết (code) một lần, thực thi khắp nơi" ("Write Once, Run Anywhere" - WORA)
- Mã nguồn được biên dịch thành bytecode rồi được thông dịch bởi JVM



33

4.3 Mô hình dịch của Java (3)

❖ Máy ảo Java (Java Virtual Machine):

- Máy ảo Java là trái tim của ngôn ngữ Java
- Tạo ra môi trường bên trong để thực thi lệnh:
 - Nạp các file .class
 - Quản lý bộ nhớ
 - Dọn "rác"
- Trình thông dịch "Just In Time - JIT"
 - Chuyển tập lệnh bytecode thành mã máy cụ thể cho từng loại CPU.



34

4.4. Tính năng của Java

- ❖ Đơn giản, hướng đối tượng và quen thuộc.
- ❖ Mạnh mẽ và an toàn.
- ❖ Kiến trúc trung lập và di động.
- ❖ Thực thi với hiệu suất cao.
- ❖ Thông dịch, đa luồng và động.
- ❖ Dễ sử dụng cho người dùng Java.

4.4. Tính năng của Java (2)

❖ Đơn giản

- Từ khóa
 - Java có 50 từ khóa
 - So với Cobol hay VB có tới hàng trăm từ khóa
 - Có ý nghĩa đặc biệt trong ngôn ngữ
 - Được sử dụng để viết các câu lệnh

❖ Hướng đối tượng

- Java hỗ trợ phát triển phần mềm bằng cách sử dụng khái niệm "đối tượng"
- Phần mềm được phát triển sử dụng Java bao gồm các lớp và các đối tượng

4.4. Tính năng của Java (3)

❖ Mạnh mẽ

- Thư viện lớp: Hàng trăm lớp được viết trước với nhiều các phương thức tiện ích.
- Java sử dụng mô hình con trỏ không cho phép truy cập trực tiếp vào bộ nhớ; bộ nhớ không thể ghi đè.

❖ An toàn

- Java authentication dựa vào các phương pháp mã hóa khóa công khai
- Mô hình con trỏ Java bảo vệ dữ liệu riêng tư trong các đối tượng và ngăn các ứng dụng chưa được authorized không được phép truy cập cấu trúc dữ liệu

4.4. Tính năng của Java (4)

4.4. Tính năng của Java (4)

❖ Kiến trúc trung lập, di động

- Hỗ trợ nhiều platform, "Write Once, Run Anywhere"

❖ Network capable

- Hỗ trợ phát triển các ứng dụng trong môi trường mạng
- Java hỗ trợ phát triển các ứng dụng phân tán
- Lý tưởng cho các ứng dụng Web

❖ Thực thi với hiệu suất cao

4.4. Tính năng của Java (5)

❖ Thông dịch

- Chương trình nguồn *.java được biên dịch thành *.class và sau đó sẽ được thông dịch thành mã máy

❖ Đa luồng (Multi-threaded)

- Cho phép chương trình của bạn chạy nhiều hơn một tác vụ tại cùng một thời điểm.

❖ Khả chuyển (Portable)

- Các chương trình có thể viết và biên dịch một lần, rồi chạy trên các nền tảng khác
- Nhờ mô hình biên dịch/thông dịch
- (WORE – Write Once, Run Everywhere)

3.5. Các ứng dụng của Java

Java Card
Oracle Java Cloud Service
Java Platform, Enterprise Edition
Oracle Java Embedded
Oracle Java ME Embedded Client
Oracle Java ME Software Development Kit
Java SE
Oracle Java SE Embedded
Java TV

ƯD trên thẻ thông minh, thẻ SIM

ƯD trên Cloud

ƯD quy mô doanh nghiệp

ƯD trên thiết bị nhúng

ƯD trên thiết bị nhúng giới hạn tài nguyên

ƯD trên Windows sử dụng thiết bị nhúng

Phiên bản chuẩn, dành cho mọi ứng dụng

Phiên bản chuẩn, dành cho ứng dụng nhúng

ƯD trên TV thông minh

Nội dung

1. Công nghệ hướng đối tượng (HDT)
2. Đối tượng và lớp
3. Các nguyên lý cơ bản của LT HDT
4. Ngôn ngữ lập trình Java
5. Cài đặt môi trường lập trình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

5.1 Các bước cài đặt Java (2)

❖ Trong Windows

- JAVA_HOME = C:\Program Files\Java\jdk1.6
- PATH = ...;%JAVA_HOME%\bin;
- CLASSPATH = C:\Program Files\Java\jdk1.6\lib\.;C:\Program Files\Java\jdk1.6\include

❖ Trong Linux

- JAVA_HOME=/usr/lib/jvm/java-7-sun
- PATH=\$PATH:\$JAVA_HOME/bin



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

43

4.1 Các bước cài đặt Java

- ❖ Bước 1: Cài đặt JDK
- ❖ Bước 2: Cài đặt các biến môi trường (nếu dùng cmd)
- ❖ Bước 3: Cài trình soạn thảo hoặc IDE (Integrated Development Environment)
 - TextPad/JCreator/NetBean/Eclipse...
- ❖ Bước 4: Lập trình/Viết mã nguồn (HelloWorld.java)
- ❖ Bước 5: Dịch
 - Gõ lệnh: javac HelloWorld.java
- ❖ Bước 6: Chạy chương trình
 - Gõ lệnh: java HelloWorld.class



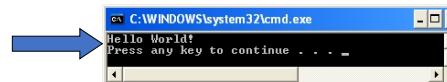
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

5.2. Chương trình ví dụ

```
// HelloWorld.java  
// Chương trình hiển thị dòng chữ "Hello World"  
public class HelloWorld {  
    public static void main(String args[]){  
        System.out.println( "Hello World!" );  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

44

5.2. Chương trình ví dụ (2)

```
import javax.swing.JOptionPane;
public class FirstDialog{
    public static void main(String[] args){
        JOptionPane.showMessageDialog(null,
            "Xin chao ban!");
        System.exit(0);
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

5.2. Chương trình ví dụ (3)

```
import javax.swing.JOptionPane;
public class HelloNameDialog{
    public static void main(String[] args){
        String result;
        result = JOptionPane.showInputDialog("Hay nhap ten ban:");
        JOptionPane.showMessageDialog(null,
            "Xin chao " + result + "!");
        System.exit(0);
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

Bài tập

Bài 1. Cài đặt Java.

Bài 2. Cài đặt một trình IDE nào đó .

Bài 3. Chạy lại 3 ví dụ trong phần 4.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài 2: Cú pháp Java cơ bản

1

Mục tiêu bài học

- ❖ Nắm được quy định cơ bản về định danh, câu lệnh, chú thích, và biến trong Java
- ❖ Sử dụng thành thạo các kiểu dữ liệu nguyên thủy trong Java
- ❖ Nắm được các loại toán tử, các cấu trúc điều khiển, và cấu trúc dữ liệu kiểu mảng trong Java
- ❖ Hiểu được ý nghĩa ngôn ngữ mô hình hóa thống nhất UML, biết các loại biểu đồ thông dụng



2

Bước 1 - Truy cập trang <https://www.udacity.com/course/java-programming-basics--ud282>

Bước 2: Đăng ký tài khoản (free)

Bước 3: Nhấn nút START FREE COURSE

3

Bài giảng e-learning

- ❖ Trong khóa học Java Programming Basics, SV học theo các bài 1, 2, và 4.
 - Lesson 1: Variables and Data Types
 - Lesson 2: Control Flow and Conditionals
 - Lesson 3: Functions (sẽ trình bày ở các bài giảng sau)
 - Lesson 4: Loops
 - Lesson 5: IntelliJ and Debugging (tham khảo)



4

Nội dung

1. Cơ bản về Java
2. Giới thiệu về UML



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1. Cơ bản về Java

1.1. Các khái niệm cơ bản

- 1.2. Biến
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử
- 1.5. Cấu trúc điều khiển
- 1.6. Mảng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

1. Cơ bản về Java

- 1.1. Các khái niệm cơ bản
- 1.2. Biến
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử
- 1.5. Cấu trúc điều khiển
- 1.6. Mảng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Định danh

❖ Định danh:

- Xâu ký tự thể hiện tên các biến, các phương thức, các lớp và nhãn
- là duy nhất trong chương trình

❖ Quy định với định danh hợp lệ (bắt buộc tuân thủ)

- Gồm các ký tự có thể là chữ cái, chữ số, ký tự '\$' hoặc '_'
- Không được phép:**
 - Bắt đầu bởi một chữ số
 - Trùng với từ khóa
 - Chưa dấu cách
- Phân biệt chữ hoa chữ thường**
 - Yourname, yourname, YourName và yourName là 4 định danh khác nhau



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8



Định danh (2)

- ❖ Quy ước với định danh - naming convention (Quy ước: không bắt buộc, nhưng nên làm theo)
 - Phải mang tính gợi nhớ
 - Ví dụ: nên dùng định danh "bookPrice" hơn là "bp" để lưu thông tin về giá 1 quyển sách
 - Bắt đầu bằng chữ cái
 - Gói (package): tất cả sử dụng chữ thường
 - theexample
 - Lớp (Class): viết hoa chữ cái đầu tiên trong các từ ghép lại
 - TheExample
 - Phương thức/thuộc tính (method/field): Bắt đầu bằng chữ thường, viết hoa chữ cái đầu tiên trong các từ còn lại
 - theExample
 - Hằng (constants): Tất cả viết hoa
 - THE_EXAMPLE



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

Câu lệnh

- ❖ Các câu lệnh kết thúc bởi dấu ;
- ❖ Nhiều lệnh có thể viết trên một dòng
- ❖ Một câu lệnh có thể viết trên nhiều dòng
 - Ví dụ:

```
System.out.println(  
    "This is part of the same line");
```

```
a=0; b=1; c=2;
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

Các từ khóa

- ❖ Người lập trình không được phép sử dụng các từ khóa như một định danh
- ❖ Literals:
 - null true false
- ❖ Từ khóa (keyword):
 - abstract assert boolean break byte case catch char class continue default do double else extends final finally float for if implements import instanceof int interface long native new package private protected public return short static strictfp super switch synchronized this throw throws transient try void volatile while
- ❖ Từ dành riêng (reserved word):
 - byvalue cast const future generic goto inner operator outer rest var volatile



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

Chú thích trong Java

- ❖ Java hỗ trợ ba kiểu chú thích như sau:
 - // Chú thích trên một dòng
 - // Không xuống dòng
 - /* Chú thích một đoạn */
 - /** Javadoc * chú thích dạng Javadoc */
- ❖ Chú thích dùng để mô tả thêm về mã nguồn (source code). Trình thông dịch sẽ bỏ qua các chú thích này.

```
/* This is a simple Java program.  
FileName : "HelloWorld.java". */  
class HelloWorld  
{  
    // Your program begins with a call to main().  
    // Prints "Hello, World" to the terminal window.  
    public static void main(String args[])  
    {  
        System.out.println("Hello, World");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

11

3

1. Cơ bản về Java

- 1.1. Các khái niệm cơ bản
- 1.2. Biến** (*Tham khảo Lesson 1 – Session 6*)
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử
- 1.5. Cấu trúc điều khiển
- 1.6. Mảng



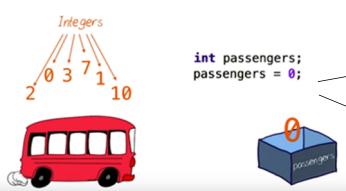
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Khai báo biến

- ❖ Biến khi dùng phải được khai báo tên (định danh) và gán cho một kiểu dữ liệu (số, ký tự, văn bản, đối tượng, v.v.)
- ❖ Các biến đơn cần phải được khởi tạo trước khi sử dụng



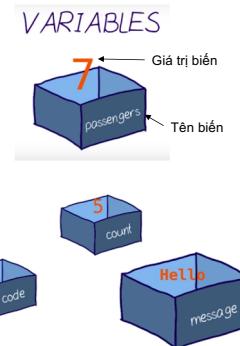
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Khái niệm biến

- ❖ Biến giống như 1 chiếc hộp trong bộ nhớ, chứa giá trị cho 1 đại lượng nào đó
 - Biến có tên không thay đổi
 - Biến được gán 1 giá trị, có thể thay đổi trong khi chạy
- ❖ Biến có thể chứa các giá trị kiểu số, ký tự, văn bản, hay đối tượng
 - và kiểu giá trị này của biến cũng không thay đổi, gọi là kiểu dữ liệu của biến



<https://www.youtube.com/watch?v=TGw5szvZk88>



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Khai báo biến (2)

- ❖ Có thể kết hợp khai báo và khởi tạo cùng một lúc.
- ❖ Sử dụng toán tử = để gán (bao gồm cả khởi tạo)
- ❖ Ví dụ:

Declaring *Initializing*
`int price = 0;
int speed = 100;
int stockPrice = 75;`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

Sử dụng biến

```
int passengers;  
passengers = 0;
```



```
passengers = passengers + 5;
```



```
passengers = passengers - 3;
```



```
System.out.println(passengers);
```

Lệnh in ra giá trị hiện tại của biến
passengers (không có "" quanh tên biến)
Nếu passengers chưa khởi tạo, sẽ báo lỗi

Phạm vi sử dụng của biến (2)

```
boolean isLightGreen = ?; //true or false  
  
if(isLightGreen) {  
    //traffic light is green  
    double carSpeed = 100; //in km/hr  
    System.out.println("Drive!");  
    System.out.println("Speed is: " + carSpeed);  
}
```

Block of code where
a variable can be used

A set of curly braces
defines a variable scope

{ scope }

isLightGreen
scope

carSpeed
scope

Phạm vi sử dụng của biến

❖ Phạm vi của biến là vùng chương trình mà trong đó biến có thể được tham chiếu đến, có thể sử dụng được.

❖ Phạm vi hoạt động (scope) của các biến cho phép xác định các nguyên lý của tạo biến, sử dụng biến và giải phóng biến

❖ Phân loại:

- Biến toàn cục: phạm vi trong cả chương trình
- Biến cục bộ: được khai báo trong một phương thức/khoi lệnh thì chỉ có thể truy cập trong phương thức/khoi lệnh đó.

1. Cơ bản về Java

1.1. Các khái niệm cơ bản

1.2. Biến

Tham khảo Lesson 1 - Session 16, 12, 13

1.3. Các kiểu dữ liệu cơ bản

1.4. Toán tử

1.5. Cấu trúc điều khiển

1.6. Mảng

Các kiểu dữ liệu trong Java

- ❖ Trong Java kiểu dữ liệu được chia thành hai loại:
 - Kiểu dữ liệu nguyên thủy (primitive)
 - Số nguyên (integer)
 - Số thực (float)
 - Ký tự (char)
 - Giá trị logic (boolean)
 - Kiểu dữ liệu tham chiếu (reference)
 - Mảng (array)
 - Đối tượng (object)
- ❖ Kích thước của các kiểu dữ liệu nguyên thủy được định nghĩa bởi JVM. Chúng giống nhau trên tất cả các platform
- ❖ Cần cân bằng giữa nhu cầu lưu trữ (độ lớn có thể của giá trị) và việc tiết kiệm bộ nhớ (không dư thừa ô nhớ)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

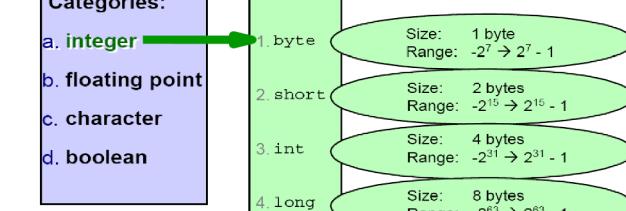
(1) Integer int maxInt = 2147483647;
(2) Long long muchMore = 2147483647*1000000;



Số nguyên

- ❖ Số nguyên có dấu
- ❖ Khởi tạo với giá trị 0

Categories:
a. integer
b. floating point
c. character
d. boolean



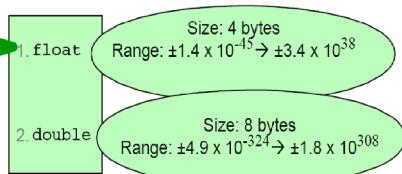
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

Số thực

- ❖ Khởi tạo với giá trị 0.0

Categories:
a. integer
b. floating point
c. character
d. boolean



double fraction = 99.275;



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

Ký tự

- ❖ Ký tự Unicode không dấu, được đặt giữa hai dấu nháy đơn
- ❖ 2 cách gán giá trị:
 - Sử dụng các chữ số trong hệ 16: char uni = '\u05D0';
 - Sử dụng ký tự: char a = 'A';
- ❖ Giá trị mặc định là giá trị zero (\u0000)

Categories:
a. integer
b. floating point
c. character
d. boolean



char answer = 'b';
char three = '3';
char ten = '\u0010';

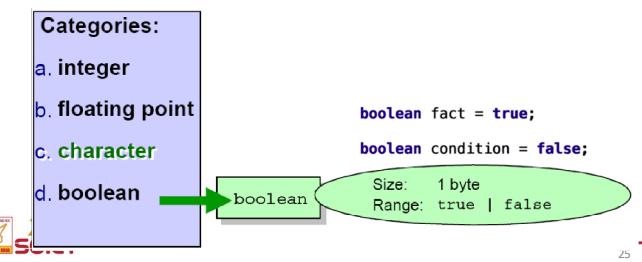


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

Giá trị logic

- ❖ Giá trị boolean được xác định rõ ràng trong Java
 - Một giá trị int không thể sử dụng thay cho giá trị boolean
 - Có thể lưu trữ giá trị hoặc true hoặc false
- ❖ Biến boolean được khởi tạo là false



25

Giá trị hằng (literal)

- ❖ Literal là một giá trị của các kiểu dữ liệu nguyên thủy và xâu ký tự.

Gồm 5 loại:

- integer
- floating point
- boolean
- character
- String

Literals

integer.....	7
floating point...	7.0f
boolean.....	true
character.....	'A'
string.....	"A"

Hằng số nguyên

- ❖ Hệ cơ số 8 (Octals) bắt đầu với chữ số 0
 - $032 = 011\ 010(2) = 16 + 8 + 2 = 26(10)$
- ❖ Hệ cơ số 16 (Hexadecimals) bắt đầu với 0 và ký tự x
 - $0x1A = 0001\ 1010(2) = 16 + 8 + 2 = 26(10)$
- ❖ Kết thúc bởi ký tự "L" thể hiện kiểu dữ liệu long
 - 26L
- ❖ Ký tự hoa, thường cho giá trị bằng nhau
 - 0xa , 0xA , 0X1a , 0X1A đều có giá trị 26 trong hệ decimal

Hằng số thực

- ❖ float kết thúc bằng ký tự f (hoặc F)
 - 7.1f
- ❖ double kết thúc bằng ký tự d (hoặc D)
 - 7.1D
- ❖ e (hoặc E) được sử dụng trong dạng biểu diễn khoa học:
 - 7.1e2
- ❖ Một giá trị thực mà không có ký tự kết thúc đi kèm sẽ có kiểu là double
 - 7.1 giống như 7.1d

Hằng boolean, ký tự và xâu ký tự

❖ boolean:

- true
- false

❖ Ký tự:

- Được đặt giữa 2 dấu nháy đơn
- Ví dụ: 'a', 'A' hoặc '\uffff'

❖ Xâu ký tự:

- Được đặt giữa hai dấu nháy kép
- Ví dụ: "Hello world", "Xin chao ban", ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

Escape sequence

❖ Các ký tự điều khiển nhấn phím

- \b backspace
- \f form feed
- \n newline
- \r return (về đầu dòng)
- \t tab

❖ Hiển thị các ký tự đặc biệt trong xâu

- \" quotation mark
- \' apostrophe
- \\ backslash



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

Chuyển đổi kiểu dữ liệu (Casting)

❖ Java là ngôn ngữ định kiểu chặt

- Gán sai kiểu giá trị cho một biến có thể dẫn đến các lỗi biên dịch hoặc các ngoại lệ của JVM

❖ JVM có thể ngầm định chuyển từ một kiểu dữ liệu hẹp sang một kiểu rộng hơn

❖ Để chuyển sang một kiểu dữ liệu hẹp hơn, cần phải định kiểu rõ ràng.

```
int a, b;  
short c;  
a = b + c;
```

```
int d;  
short e;  
e = (short)d;
```

```
double f;  
long g;  
f = g;  
g = f; //error
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

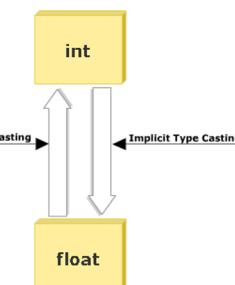
Chuyển đổi kiểu dữ liệu (2)

❖ Chuyển đổi kiểu sẽ được thực hiện tự động nếu không xảy ra mất mát thông tin

- byte → short → int → long → float → double

❖ Lưu ý: ép kiểu từ short về char, từ byte về char và ngược lại đều phải ép kiểu tường minh

❖ Ép kiểu trực tiếp (explicit cast) được yêu cầu nếu có "nguy cơ" giảm độ chính xác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

Ví dụ - chuyển đổi kiểu

```
long p = (long) 12345.56; // p sẽ nhận giá trị  
12345  
int q = p; // không hợp lệ dù kiểu int có thể  
lưu giá trị 12345  
char c = 't';  
int j = c; // hợp lệ, tự động chuyển đổi  
short k = c; // không hợp lệ, phải ép kiểu tường  
minh  
short k = (short) c; // hợp lệ  
float f = 12.35; // Báo lỗi do 12.35 là hằng  
double  
float f = 0.0; // Báo lỗi do 0.0 là hằng double  
float f = 0;  
long l = 999999999999; //Báo lỗi: The literal  
999999999999 of type int is out of range  
short k = 99999999; // Báo lỗi: Type mismatch:  
cannot convert from int to short
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

1. Cơ bản về Java

- 1.1. Các khái niệm cơ bản
- 1.2. Biến
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử** *Tham khảo Lesson 1 - Session 18*
- 1.5. Cấu trúc điều khiển
- 1.6. Mảng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

Ví dụ SV tự thử và kiểm tra kết quả

```
short i = 6, j=7;
```

```
i = i + j;
```

```
i += j;
```

```
short i, j = 5;
```

```
int n = 6;
```

```
i = (short)n + j;
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

Toán tử (Operators)

- ❖ Kết hợp các giá trị đơn hoặc các biểu thức con thành những biểu thức mới, phức tạp hơn và có thể trả về giá trị.
- ❖ Java cung cấp nhiều dạng toán tử sau:

- Toán tử số học
- Toán tử bit, toán tử quan hệ
- Toán tử logic
- Toán tử gán
- Toán tử một ngôi

✓ A = B + C
✓ Z = Y * Y
✓ Result = (A+B) > (C+D)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

37

9

Toán tử (2)

❖ Toán tử số học

- +, -, *, /, %

❖ Toán tử bit

- AND: &, OR: |, XOR: ^, NOT: ~
- Dịch bit: <<, >>

❖ Toán tử quan hệ

- ==, !=, >, <, >=, <=

❖ Toán tử logic

- &&, ||, !



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

Toán tử “/”

```
int i = 10/3;
float f0 = 10;
float f1 = (float) 10/3;
float f2 = 10/3;
float f3 = f0/3;
System.out.println(i);
System.out.println(f1) //3
System.out.println(f2) //3.3333333
System.out.println(f3) //3.0
                           //3.3333333
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

Toán tử (3)

❖ Toán tử một ngôi

- Đảo dấu: +, -
- Tăng giảm 1 đơn vị: ++, --
- Phủ định một biểu thức logic: !

❖ Toán tử gán

- =, +=, -=, %= tương tự với >>, <<, &, |, ^



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

Thứ tự ưu tiên của toán tử

❖ Cho biết toán tử nào thực hiện trước

❖ Được xác định bởi các dấu ngoặc đơn hoặc theo ngầm định như sau (ưu tiên từ trên xuống thực hiện trước):

- 1.Toán tử [] . ()
- 2.Toán tử x++ x--
- 3.Toán tử một ngôi: ++x --x +x -x ~ !
- 4.Toán tử khởi tạo, toán tử chuyển kiểu: new (type)x
- 5.Nhân, chia: * / %
- 6.Cộng, trừ: + -
- 7.Dịch bit: << >> >>> (unsigned shift)
- 8.So sánh: < > <= >= instanceof
- 9.So sánh bằng == !=
- 10.Toán tử bit AND: &
- 11.Toán tử bit OR: ^
- 12.Toán tử bit XOR: -
- 13.Toán tử logic AND: &&
- 14.Toán tử logic OR: ||
- 15.Toán tử điều kiện: (ternary) ?:
- 16.Toán tử gán: = *= /= %= += -= >>= <<= >>>= &= ^= |=



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

10

Thứ tự ưu tiên của toán tử - Ví dụ 1

```
double paid = 10;  
double change = 3.25;  
double tip = (paid-change)*0.2;
```

$$\begin{aligned} d &= (\text{paid} - \text{change}) * 0.2 \\ 1.35 &= (10 - 3.25) * 0.2 \\ 9.35 & \end{aligned}$$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

Thứ tự ưu tiên của toán tử - Ví dụ 2

```
int i;  
System.out.println(i=5); // 5  
System.out.println(i+=4); // 9  
System.out.println(i++); // 9  
System.out.println(--i); // 9
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

1. Cơ bản về Java

- 1.1. Các khái niệm cơ bản
- 1.2. Biến
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử

1.5. Cấu trúc điều khiển *Tham khảo Lesson 2 – Session 1..16*

- 1.6. Mảng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

1.5. Cấu trúc điều khiển

- ❖ Là các cấu trúc lệnh nhằm chỉ định cho chương trình thực hiện các câu lệnh/đoạn lệnh khác nhau, tùy theo từng điều kiện nào đó.
- ❖ 2 loại cấu trúc điều khiển:
 - Câu lệnh điều kiện
 - Lệnh if – else,
 - Lệnh switch – case
 - Câu lệnh lặp
 - Vòng lặp for
 - Vòng lặp while
 - Vòng lặp do – while



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

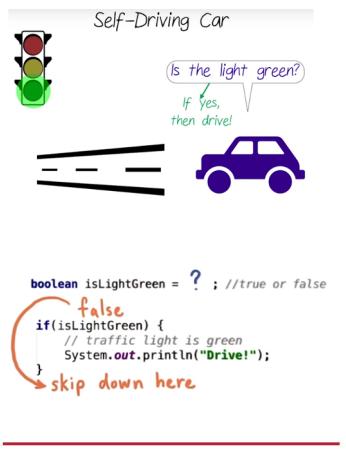
Lệnh if

❖ Cú pháp

```
if (dieu_kien) {
    cac_cau_lenh;
}
```

❖ Nếu biểu thức điều kiện dieu_kien (có kiểu boolean) nhận giá trị true thì thực hiện khối lệnh cac_cau_lenh;

```
boolean isLightGreen = ?; //true or false
if(isLightGreen) {
    // traffic light is green
    System.out.println("Drive!");
}
```



47

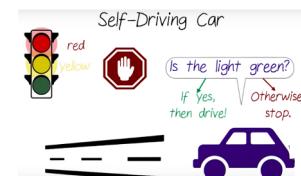
Lệnh if - else

❖ Cú pháp

```
if (dieu_kien) {
    cac_cau_lenh_1;
} else {
    cac_cau_lenh_2;
}
```

❖ Nếu biểu thức điều kiện (kiểu boolean) nhận giá trị true thì thực hiện khối lệnh cac_cau_lenh_1, là false thì thực hiện khối lệnh cac_cau_lenh_2.

```
boolean isLightGreen = ?; //true or false
if(isLightGreen) {
    // traffic light is green
    System.out.println("Drive!");
} else {
    // light is NOT green
    System.out.println("Stop.");
}
```



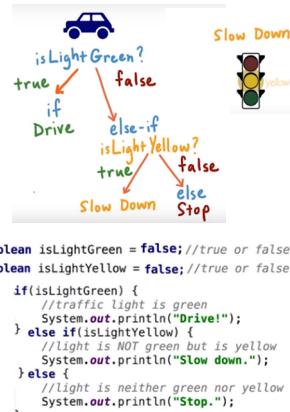
48

Lệnh else-if

❖ Cú pháp

```
if (dieu_kien_1) {
    cac_cau_lenh_1;
} else_if (dieu_kien_2) {
    cac_cau_lenh_2;
} else_if (dieu_kien_3) {
    cac_cau_lenh_3;
} else {
    cac_cau_lenh_n;
}
```

❖ Có thể có nhiều else-if, chỉ có 1 else tối đa



49

Biểu thức điều kiện

❖ Toán tử so sánh

Expression	Value
int x = 10;	true
3 < 5	false
3 > 5	false
7 <= 6	false
x >= 10	true
x == 9	false
equality check	
x != 9	true
NOT equal	

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

Biểu thức điều kiện (2)

❖ Toán tử logic

Three main logical operators:

- 1) AND $3 < 5 \ \&\& \ 2 > 15 \rightarrow \text{false}$
- 2) OR $3 < 5 \ || \ 2 > 15 \rightarrow \text{true}$
- 3) NOT $!(3 < 5) \rightarrow \text{false}$
turns a value into its opposite

Using multiple logical operators
 $\&\&$ will evaluate first, then $||$

$\text{false} \ \&\& \ \text{true} \ || \ \text{true} \rightarrow \text{true}$
 $\text{false} \ \&\& (\text{true} \ || \ \text{true}) \rightarrow \text{false}$

Museum discount cases



```
if( age <= 15 || age > 60 || isStudent ) {
```

```
    1   2   3
```

$\text{ticket} = \$5$



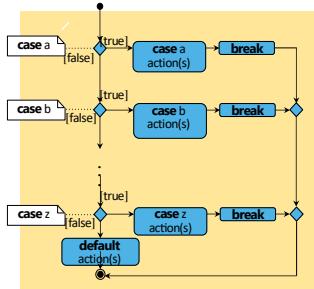
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

51

Lệnh switch - case

- ❖ Kiểm tra một biến đơn với nhiều giá trị khác nhau và thực hiện trường hợp tương ứng
 - break: Thoát khỏi lệnh switch-case
 - default kiểm soát các giá trị nằm ngoài các giá trị case:



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

Ví dụ - Kiểm tra số chẵn – lẽ

```
class CheckNumber
{
    public static void main(String args[])
    {
        int num = 10;
        if (num % 2 == 0)
            System.out.println (num+ " là số chẵn");
        else
            System.out.println (num + " là số lẻ");
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

Ví dụ - Lệnh switch - case (1)

```
public class Test {
    public static void main(String args[]) {
        int i = 2;

        switch (i) {
            case 1:
                System.out.println("1");
            case 2:
                System.out.println("2");
            case 3:
                System.out.println("3");
        }
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

13

Ví dụ - Lệnh switch - case (2)

```
switch (day) {
    case 0:
    case 1:
        rule = "weekend";
        break;
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
        rule = "weekday";
        break;
    default:
        rule = "error";
}
if (day == 0 || day == 1) {
    rule = "weekend";
} else if (day > 1 && day <7) {
    rule = "weekday";
} else {
    rule = error;
}
```

Bài tập: Tính số ngày trong tháng

- ❖ Input: Năm, tháng
- ❖ Output: số ngày trong tháng của năm đó
- ❖ Yêu cầu: sử dụng lệnh switch-case

❖ Gợi ý:

- Tháng 1, 3, 5, 7, 8, 10, 12: 31 ngày
- Tháng 4, 6, 9, 11: 30 ngày
- Riêng tháng 2:
 - Năm thường: 28 ngày
 - Năm nhuận: 29 ngày (năm nhuận là "năm chia hết cho 4 và không chia hết cho 100", hoặc là "năm chia hết cho 400")

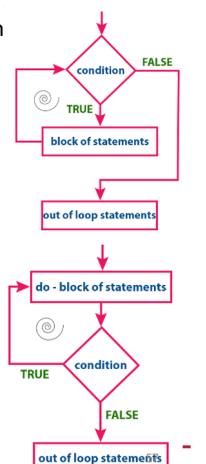
d. Vòng lặp while và do while

- ❖ Thực hiện một câu lệnh hoặc một khối lệnh
- ❖ khi điều kiện vẫn nhận giá trị true

```
while (condition) {
    // code block to be executed
}

do {
    // code block to be executed
} while (condition);
```

- ❖ while() thực hiện 0 hoặc nhiều lần
- ❖ do...while() thực hiện ít nhất một lần



Ví dụ - Vòng lặp while

```
int numOfWorkings = 5;
int i = 1;           ← (1) loop counter
while (i <= numOfWorkings) { ← (2) loop condition
    System.out.println("Warning");
    i++;               ← (3) loop increment
}
```

Print output

```
Warning
Warning
Warning
Warning
Warning
```

Chú ý: Cần tránh vòng lặp vô tận!

i	i<=numOfWorkings
1	1<=5 (true)
2	2<=5 (true)
3	3<=5 (true)
4	4<=5 (true)
5	5<=5 (true)
6	6<=5 (false)

Ví dụ - Vòng lặp while (2)

```
class WhileDemo{  
    public static void main(String args[]){  
        int a = 5,fact = 1;  
        while (a >= 1){  
            fact *=a;  
            a--;  
        }  
        System.out.println("The Factorial of 5 is"+fact);  
    }  
}
```

Kết quả: "The factorial of 5 is 120" được hiển thị.

Viết thay lệnh while bằng lệnh do-while ?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

Ví dụ - Vòng lặp for

```
class ForDemo {  
    public static void main(String args[]){  
        int i=1, sum=0;  
        for (i=1;i<=10;i+=2){  
            sum+=i;  
        }  
        System.out.println("Sum of first five odd numbers is" + sum);  
    }  
}
```

Kết quả: "Sum of first five odd numbers is 25" được hiển thị.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

Vòng lặp for

❖ Cú pháp:

```
for (start_expr; test_expr; increment_expr){  
    // code to execute repeatedly  
}
```

❖ Ví dụ:

```
loop counter      loop condition      loop increment  
(1)             (2)                  (3)  
for(int i = 1; i <= numWarnings ; i++){  
    System.out.println("Warning");  
}
```

❖ 3 biểu thức (1) (2) (3) đều có thể vắng mặt (thay bằng lệnh tương ứng trong khối lệnh)

❖ Có thể khai báo biến trong câu lệnh for

- Thường sử dụng để khai báo một biến đếm
- Thường khai báo trong biểu thức "start"
- Phạm vi của biến giới hạn trong vòng lặp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

Vòng lặp for và while

❖ Các câu lệnh for và while cung cấp chức năng tương đương nhau

❖ Các cấu trúc lặp thường được sử dụng trong các tình huống khác nhau

- while được sử dụng cho lặp từ đầu đến cuối
- for được sử dụng để lặp với số vòng lặp xác định

```
int sum = 0;  
for (int index = 1;index <= 10;index++)  
{  
    sum += index;  
}
```

```
int sum = 0;  
int index = 1;  
while (index <= 10) {  
    sum += index;  
    index++;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

61

15

Các lệnh thay đổi cấu trúc điều khiển

❖ break

- Có thể được sử dụng để thoát ra ngoài câu lệnh switch
- Kết thúc vòng lặp for, while hoặc do...while
- Có hai dạng:
 - Gắn nhãn: Tiếp tục thực hiện câu lệnh tiếp theo sau vòng lặp được gắn nhãn
 - Không gắn nhãn: Thực hiện câu lệnh tiếp theo bên ngoài vòng lặp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

63

Ví dụ - break và continue

```
public int myMethod(int x) {  
    int sum = 0;  
    outer: for (int i=0; i<x; i++) {  
        inner: for (int j=i; j<x; j++) {  
            sum++;  
            if (j==1) continue;  
            if (j==2) continue outer;  
            if (i==3) break;  
            if (j==4) break outer;  
        }  
    }  
    return sum;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

65

4.5. Các lệnh thay đổi cấu trúc điều khiển (2)

❖ continue

- Có thể được sử dụng cho vòng lặp for, while hoặc do...while
- Bỏ qua các câu lệnh còn lại của vòng lặp hiện thời và chuyển sang thực hiện vòng lặp tiếp theo.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

64

64

1. Cơ bản về Java

- 1.1. Các khái niệm cơ bản
- 1.2. Biến
- 1.3. Các kiểu dữ liệu cơ bản
- 1.4. Toán tử
- 1.5. Cấu trúc điều khiển
- 1.6. Mảng**



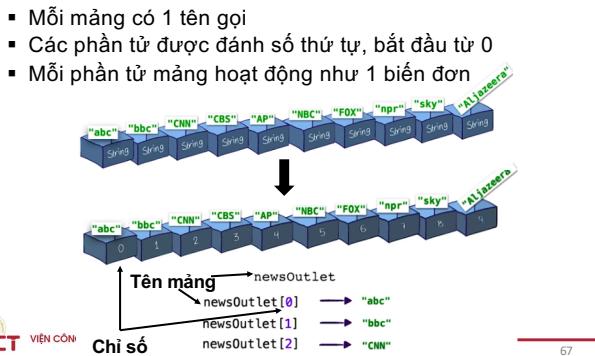
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

16

Khái niệm Mảng (array)

- Dùng để lưu một tập hợp hữu hạn các phần tử cùng kiểu (nguyên thuỷ hoặc đối tượng), liền kề nhau trong bộ nhớ.



67

Khai báo và khởi tạo mảng

- Mảng phải khai báo trước khi sử dụng. Kích thước của một mảng sau khai báo sẽ không thể thay đổi
- Cú pháp:

```
kieu_dulieu[] ten_mang = new
kieu_dulieu[KichThuoc_MANG];
kieu_dulieu ten_mang[] = new
kieu_dulieu[KichThuoc_MANG];
kieu_dl[] ten_mang = {ds_gia_tri_cac_ptu};
```

- Nếu không khởi tạo → tất cả các phần tử của mảng nhận giá trị mặc định tùy thuộc vào kiểu dữ liệu.

Khai báo và khởi tạo mảng

Cách khai báo	Mô tả	Cú pháp	Ví dụ
Chỉ đơn thuần khai báo mảng	Chỉ đơn thuần khai báo mảng	Datatype identifier[]	<code>char ch[];</code> khai báo mảng ký tự có tên ch
Khai báo và tạo mảng	Khai báo và cấp phát bộ nhớ cho các phần tử mảng sử dụng toán tử <code>new</code>	Datatype identifier[] = new datatype [size]	<code>char ch[] = new char [10];</code> Khai báo một mảng ch và lưu trữ 10 ký tự
Khai báo và khởi tạo các phần tử mảng	Khai báo mảng, cấp phát bộ nhớ cho nó và gán các giá trị ban đầu cho các phần tử của mảng	Datatype identifier[] = {value1, value2 ... valueN};	<code>char ch [] = {'A', 'B', 'C', 'D'};</code> khai báo mảng ch và lưu 4 chữ cái kiểu ký tự

Ví dụ - mảng

Tên của mảng (tất cả các thành phần trong mảng có cùng tên, c)	c[0]	-45
	c[1]	6
	c[2]	0
	c[3]	72
	c[4]	1543
	c[5]	-89
	c[6]	0
	c[7]	62
	c[8]	-3
	c[9]	1
	c[10]	6453
Chỉ số (truy nhập đến các thành phần của mảng thông qua chỉ số)	c[11]	78

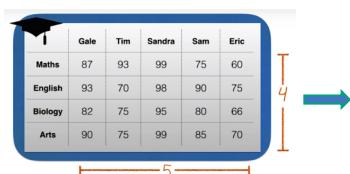
Khai báo và khởi tạo mảng

- ❖ Ví dụ:

```
int MAX = 5;  
boolean bit[] = new boolean[MAX];  
float[] value = new float[2*3];  
int[] number = {10, 9, 8, 7, 6};  
System.out.println(bit[0]); // prints "false"  
System.out.println(value[3]); // prints "0.0"  
System.out.println(number[1]); // prints "9"
```

Mảng 2 chiều

- ❖ Có thể khai báo và sử dụng một mảng nhiều chiều.
- ❖ Mảng 2 chiều giống một bảng với các dòng và cột.



2D Array					
	Gale	Tim	Sandra	Sam	Eric
Maths	87	93	99	75	60
English	93	70	98	90	75
Biology	82	75	95	80	66
Arts	90	75	99	85	70

Làm việc với mảng

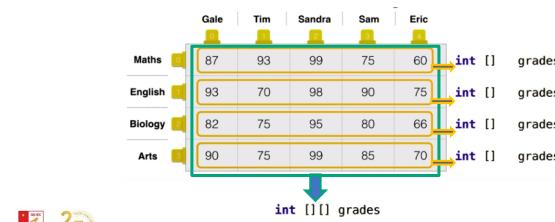
- ❖ Dùng thuộc tính .length để lấy kích thước của một mảng
- ❖ Lưu ý không truy cập vào các chỉ số không thuộc mảng, ví dụ chỉ số âm, chỉ số \geq kích thước mảng.
- ❖ Duyệt tất cả các phần tử trong mảng: dùng vòng lặp.

```
temperatures: 74 73 72 80  
int size = temperatures.length; ← 4  
System.out.println(temperatures[10]); Error!  
ArrayIndexOutOfBoundsException  
loop counter: (0, 1, 2 ...)
```

Mảng 2 chiều (2)

- ❖ Mảng 2 chiều: coi như 1 mảng của các phần tử A, mỗi phần tử A lại là 1 mảng các phần tử B.
- ❖ Khai báo mảng 2 chiều

```
kieu_dulieu[][] ten_mang;
```



Mảng 2 chiều (3)

- Truy cập phần tử trong mảng:

ten_mang [chi_so_hang] [chi_so_cot]

- Duyệt tất cả các phần tử trong mảng:

- Dùng vòng lặp lồng nhau

	Gale	Tim	Sandra	Sam	Eric
Maths	87	93	99	75	60
English	93	70	98	90	75
Biology	82	75	95	80	66
Arts	90	75	99	85	70

grades[2][1] = 75
array #2 (Biology)
item #1 (Tim)

```
for(int i=0; i<4; i++){  
    for (int j=0; j<5; j++){  
        //truy cập grades[i][j];  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

75

75

2. Giới thiệu về UML

- UML là gì
- Các biểu đồ UML cơ bản
- Ví dụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

77

77

Bài tập

- Bài tập 1: Viết chương trình tráo đổi ngẫu nhiên vị trí một dãy số cho trước

- Để lấy một số int ngẫu nhiên từ 0 đến n-1 ta dùng lệnh
 - int i = Random.nextInt(n);

- Bài tập 2: Viết chương trình sắp xếp một dãy số theo thứ tự tăng dần, dãy số được nhập từ bàn phím.

- Bài tập 3: Viết chương trình nhập chiều cao h từ bàn phím, sau đó hiển thị các tam giác hình sao có chiều cao h như dưới đây. Chú ý có kiểm tra điều kiện của h: $2 \leq h \leq 10$. Nếu h nằm ngoài đoạn trên, yêu cầu người dùng nhập lại.

- Bài tập 4: Nhập vào kích thước ô vuông $n \times n$, kiểm tra $3 \leq n \leq 8$. Hiển thị ra màn hình kết quả như ví dụ sau.

```
1   2   3   4  
12  13  14  5  
11  16  15  6  
10  9   8   7
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

76

76

Bài giảng e-learning tham khảo

- Đăng ký tài khoản trên <https://www.udacity.com>
- Theo dõi bài giảng “Software Architecture & Design”: <https://www.udacity.com/course/software-architecture-design-ud821>
- Tập trung vào Lesson 4, các bài khác tham khảo thêm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

78

78

Bài giảng e-learning

Cấu trúc bài giảng e-learning

Lesson 4: P2L1 Review UML

Diagrams

SEND FEEDBACK

SEARCH

RESOURCES

CONCEPTS

- 1. Diagrams
- 2. OMT
- 3. UML
- 4. Diagram Types
- 5. Quiz: Diagram Quiz

way of doing that is with diagrams. This course focused on UML, using UML and

0:00 / 1:05

YouTube

Chọn phụ đề nếu cần

79

79

Bài giảng e-learning tham khảo

❖ Bài giảng của Smartdraw

- <https://www.smartdraw.com/uml-diagram/>
- <https://www.smartdraw.com/use-case-diagram/>
- <https://www.smartdraw.com/activity-diagram/>
- <https://www.smartdraw.com/sequence-diagram/>
- <https://www.smartdraw.com/class-diagram/>



80

2. Giới thiệu về UML

2.1. UML là gì

2.2. Các biểu đồ UML cơ bản

2.3. Ví dụ

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

81

81

Tầm quan trọng của phân tích và thiết kế

❖ Hướng tiếp cận không có phân tích – thiết kế:

- Bắt đầu lập trình ngay khi có được yêu cầu
- Mất rất nhiều thời gian và tạo đi tạo lại nhiều mã nguồn
- Không có bất kỳ một kiến trúc nào
- Phải chịu khổ với những lỗi phát sinh



❖ Hướng tiếp cận có phân tích – thiết kế:

- Chuyển các yêu cầu của bài toán thành một bản thiết kế rõ ràng
- Tập trung vào phân tích các YÊU CẦU và thiết kế các MÔ HÌNH cho hệ thống TRƯỚC khi lập trình



82

20

Tầm quan trọng của phân tích và thiết kế (2)

- ❖ Ưu điểm của việc PTTK hệ thống:
 - Đơn giản hóa thế giới thực bằng các mô hình
 - Mô tả đúng, đồng nhất cấu trúc, cách ứng xử của HT trong suốt quá trình xây dựng
 - Đảm bảo mục đích và yêu cầu của HT được thỏa mãn trước khi xây dựng
 - Cung cấp cho người dùng, khách hàng, kỹ sư phân tích, thiết kế, kỹ sư lập trình nhiều cái nhìn khác nhau về cùng một HT
 - Ghi lại các quyết định của nhà phát triển để sử dụng sau này



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

83

Khái niệm UML

- ❖ UML: viết tắt của “Unified Modeling Language” là một Ngôn ngữ mô hình hóa được thống nhất
- ❖ UML là ngôn ngữ trực quan để:
 - trực quan hóa (visualizing)
 - đặc tả (specifying)
 - xây dựng (constructing)
 - tài liệu hóa (documenting)
- ❖ các cấu phần của một hệ thống phần mềm
- ❖ Giúp công việc phát triển được xử lý nhất quán, giảm thiểu lỗi xảy ra
 - Giúp dễ hình dung hơn cấu trúc của hệ thống
 - Hiệu quả hơn trong việc liên lạc, trao đổi



83

Lịch sử phát triển UML

- ❖ Vào năm 1994, có hơn 50 phương pháp mô hình hóa hướng đối tượng:
 - Fusion, Shlaer-Mellor, ROOM, Class-Relation, Wirfs-Brock, Coad-Yourdon, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS ...
 - Mô tả về mô hình “Meta-models” tương đồng với nhau
 - Các ký pháp đồ họa khác nhau
 - Quy trình khác nhau hoặc không rõ ràng
- Cần chuẩn hóa và thống nhất các phương pháp
- ❖ UML được 3 chuyên gia hướng đối tượng hợp nhất các kỹ thuật của họ vào năm 1994:
 - Booch91 (Grady Booch): Conception, Architecture
 - OOSE (Ivar Jacobson): Use cases
 - OMT (Jim Rumbaugh): Analysis

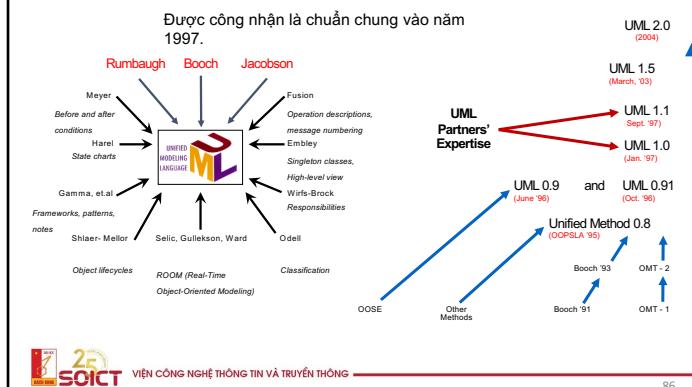


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

85

Lịch sử phát triển UML (2)

- ❖ UML là ngôn ngữ hợp nhất các mô hình khác nhau



85

86

Làm việc với UML

- ❖ Các mô hình UML có thể kết nối trực tiếp với rất nhiều ngôn ngữ lập trình.
 - Ánh xạ sang Java, C++, Visual Basic...
 - Các bảng trong RDBMS hoặc kho lưu trữ trong OODBMS
 - Cho phép các kỹ nghệ xuôi (chuyển UML thành mã nguồn)
 - Cho phép kỹ nghệ ngược (xây dựng mô hình hệ thống từ mã nguồn)
- ❖ Các công cụ UML
 - Công cụ mã nguồn mở: EclipseUML, UmlDesigner, StarUML, Argo UML, ...
 - Công cụ thương mại: Enterprise Architect, IBM Rational Software Architect, Microsoft Visio, Visual Paradigm for UML, SmartDraw...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

87

2.1. Biểu đồ UML

- ❖ Biểu đồ:
 - là các hình vẽ bao gồm các ký hiệu phần tử mô hình hóa
 - minh họa một thành phần cụ thể hay một khía cạnh cụ thể của hệ thống.
- ❖ Một mô hình hệ thống thường có nhiều loại biểu đồ, mỗi loại gồm nhiều biểu đồ khác nhau.
- ❖ Một biểu đồ là một thành phần của một hướng nhìn cụ thể
- ❖ Một số loại biểu đồ có thể là thành phần của nhiều hướng nhìn khác nhau
- ❖ UML thế hệ 2 có tới 13-14 loại biểu đồ. Trong một project, chỉ sử dụng những biểu đồ phù hợp nhất



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

89

2. Giới thiệu về UML

- 2.1. UML là gì
- 2.2. Các biểu đồ UML cơ bản
- 2.3. Ví dụ

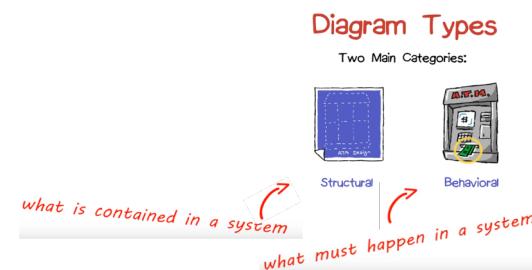


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

88

2.1. Biểu đồ UML (2)

- ❖ Phân biệt:
 - Biểu đồ cấu trúc: mô tả thành phần tĩnh, luôn có của hệ thống và mối quan hệ giữa chúng
 - Biểu đồ hành vi: mô tả cách hoạt động của hệ thống



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

90

Biểu đồ cấu trúc

- ❖ Biểu đồ cấu trúc tĩnh
 - Biểu đồ lớp (Class Diagram)
 - Biểu đồ đối tượng (Object Diagram)
 - Biểu đồ gói (Package diagram)
- ❖ Biểu đồ thực thi
 - Biểu đồ thành phần (Component Diagram)
 - Biểu đồ triển khai (Deployment Diagram)
 - Biểu đồ cấu thành (Composite Diagram)
- ❖ Biểu đồ profile (Profile Diagram)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

91

91

2. Giới thiệu về UML

- 2.1. UML là gì
- 2.2. Các biểu đồ UML cơ bản
- 2.3. Ví dụ**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

93

Biểu đồ hành vi

- ❖ Biểu đồ use case (Use Case Diagram)
- ❖ Biểu đồ hoạt động (Activity Diagram)
- ❖ Biểu đồ tương tác
 - Biểu đồ tổng quát (Interaction overview diagram)
 - Biểu đồ trình tự (Sequence Diagram)
 - Biểu đồ giao tiếp/cộng tác (Communication/Collaboration Diagram)
- ❖ Biểu đồ trạng thái (State Diagram)
- ❖ Biểu đồ thời gian (Timing Diagram)



Behavioral



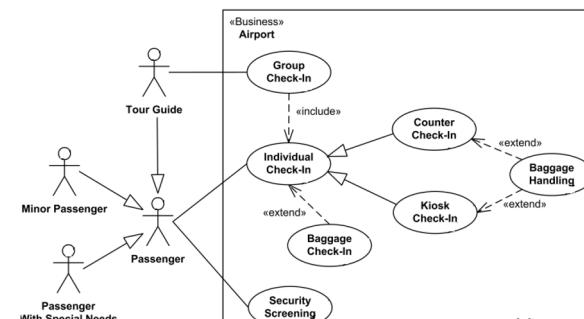
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

92

92

Biểu đồ Use case

- ❖ Mô hình chức năng hệ thống với các tác nhân và use case



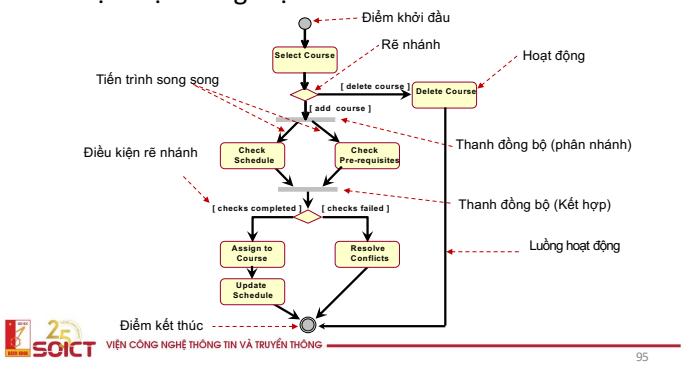
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

94

94

Biểu đồ hoạt động

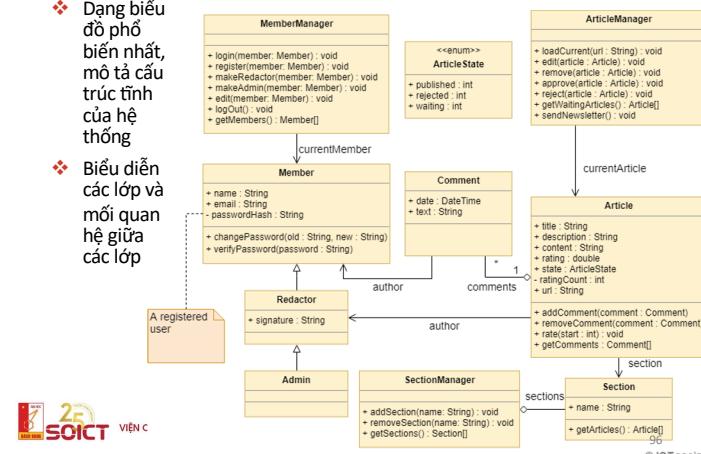
- ❖ Biểu đồ hoạt động biểu diễn chuỗi các hoạt động hoặc luồng điều khiển có thứ tự của hệ thống thực hiện trong một use case



95

Biểu đồ lớp

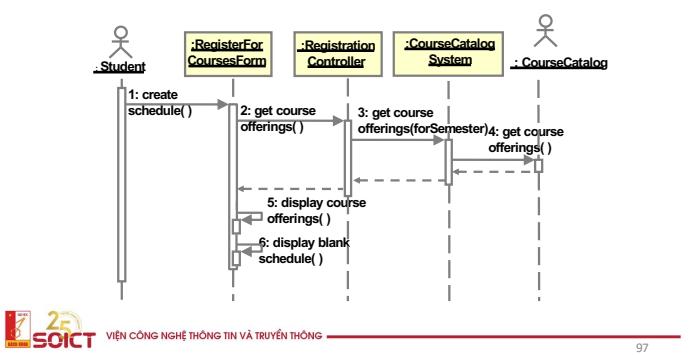
- ❖ Dạng biểu đồ phổ biến nhất, mô tả cấu trúc tĩnh của hệ thống
- ❖ Biểu diễn các lớp và mối quan hệ giữa các lớp



96

Biểu đồ tuần tự

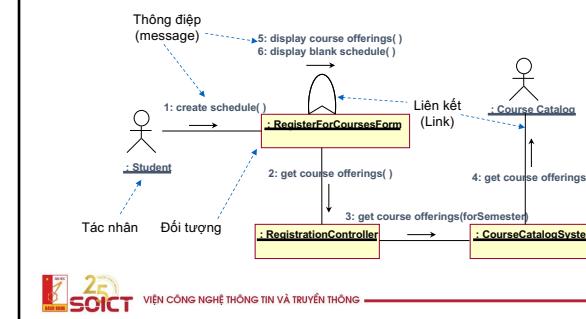
- ❖ Là một loại biểu đồ tương tác, biểu diễn trình tự trao đổi thông điệp giữa các đối tượng theo thời gian trong một use case



97

Biểu đồ giao tiếp

- ❖ Cũng là một loại biểu đồ tương tác, nhưng nhấn mạnh vào việc tổ chức các đối tượng tham gia vào tương tác hơn là trình tự trao đổi thông điệp giữa các đối tượng



98



Bài 3: Trùu tượng hóa và Đóng gói

1

Mục tiêu bài học

- ❖ Tìm hiểu bản chất, vai trò của trùu tượng hóa
 - Khái niệm, các góc nhìn, so sánh lớp và đối tượng
- ❖ Tìm hiểu về Đóng gói
 - Khái niệm đóng gói, che giấu dữ liệu
 - Chỉ định truy cập
 - Phương thức getter/setter
- ❖ Tìm hiểu cách xây dựng lớp, gói
 - Xây dựng lớp trong Java
 - Quản lý lớp với package
 - Biểu diễn đối tượng, lớp, gói trong UML



2

Nội dung

1. Trùu trưng hóa
2. Xây dựng lớp
3. Đóng gói và che giấu dữ liệu



3

Nội dung

1. Trùu trưng hóa
2. Xây dựng lớp
3. Đóng gói và che giấu dữ liệu



4

3

4

1.1 Trừu tượng hóa

- ❖ Là một trong 4 nguyên lý cơ bản của lập trình HDT.
- ❖ Là quá trình loại bỏ đi các thông tin ít quan trọng và giữ lại những thông tin quan trọng, có ý nghĩa.
- ❖ 2 loại trừu tượng hóa
 - Trừu tượng hóa điều khiển
 - Trùu tượng hóa dữ liệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

Trùu tượng hóa dữ liệu

- ❖ Trùu tượng hóa dữ liệu là một cách nhìn hoặc cách biểu diễn một thực thể chỉ bao gồm các thuộc tính liên quan trong một ngữ cảnh nào đó.
- ❖ Dựa vào các đặc điểm, thuộc tính đó để phân biệt các thực thể khác nhau trong ngữ cảnh đó.
- ❖ Góc nhìn khác nhau (bài toán khác nhau) thì đặc điểm, thuộc tính dùng để trừu tượng hóa sẽ khác nhau.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

1.1. Trùu tượng hóa (2)

- ❖ Trùu tượng hóa điều khiển:
 - Sử dụng các chương trình con (subprogram) và các luồng điều khiển (control flow)
 - Ví dụ: $a := (1 + 2) * 5$
 - Nếu không có trừu tượng hóa điều khiển, LTV phải chỉ ra tất cả các thanh ghi, các bước tính toán mức nhị phân...
- ❖ Trùu tượng hóa dữ liệu:
 - Xử lý dữ liệu theo các cách khác nhau tùy bài toán



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Ví dụ: Điện thoại Nokia



- ❖ Những thông tin về các "đối tượng" này?
 - Tất cả là điện thoại Nokia
 - Có loại nắp trượt, có loại nắp gập, có loại dạng bar
 - Một số điện thoại là dòng doanh nhân, một số dòng âm nhạc, 3G...
 - Bàn phím loại tiêu chuẩn, QWERTY hoặc không có bàn phím
 - Màu sắc, chất liệu, kích cỡ... khác nhau
 - v.v...
- ❖ Tùy bài toán, chỉ "trích rút" lấy những thông tin quan trọng, phù hợp

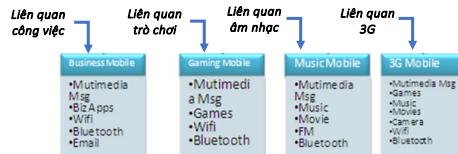


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

Ví dụ: Điện thoại Nokia (2)

- ❖ Các bài toán khác nhau, yêu cầu mô tả các tính chất khác nhau về chiếc điện thoại.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

1.2 Lớp

- ❖ Lớp (Class) là cách phân loại các đối tượng dựa trên đặc điểm chung của các đối tượng đó.
- ❖ Lớp chính là kết quả của quá trình *trừu tượng hóa dữ liệu*.
 - Lớp định nghĩa một *kiểu dữ liệu* mới, trừu tượng hóa một tập các đối tượng
 - Một đối tượng gọi là một *thể hiện* của lớp
- ❖ Lớp gồm các *phương thức* và *thuộc tính* chung của các đối tượng cùng một loại.

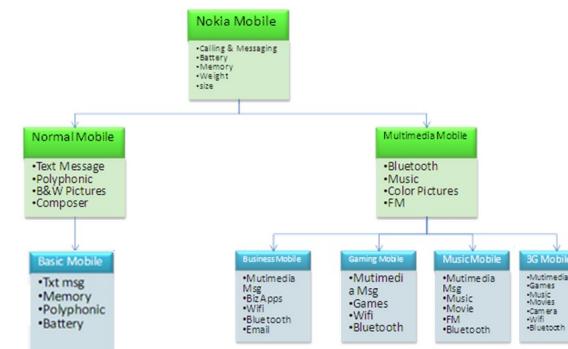


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

Ví dụ: Điện thoại Nokia (3)

- ❖ Có thể trừu tượng hóa nhiều mức.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

Thuộc tính

- ❖ Thuộc tính
 - Một thuộc tính của một lớp là một trạng thái chung được đặt tên của *tất cả* các thể hiện của lớp đó có thể có.
 - Ví dụ: Lớp Ô tô có các thuộc tính
 - Màu sắc
 - Vận tốc
- ❖ Mỗi đối tượng có bản sao các thuộc tính của riêng nó
 - Ví dụ: một chiếc Ô tô đang đi có thể có màu đen, vận tốc 60 km/h



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

Phương thức

❖ Phương thức:

- Xác định các hoạt động chung mà tất cả các thể hiện của lớp có thể thực hiện được.
 - Xác định cách một đối tượng đáp ứng lại một thông điệp
- ❖ Thông thường các phương thức sẽ hoạt động trên các thuộc tính và thường làm thay đổi các trạng thái của đối tượng.
- Bất kỳ phương thức nào cũng phải thuộc về một lớp nào đó.
 - Ví dụ: Lớp Ô tô có các phương thức
 - Tăng tốc
 - Giảm tốc



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Lớp vs. Đối tượng

Lớp	Đối tượng
Lớp là mô hình khái niệm, mô tả các thực thể	Đối tượng là sự vật thật, là thực thể thực sự
Lớp như một bản mẫu, định nghĩa các thuộc tính và phương thức chung của các đối tượng	Mỗi đối tượng có một lớp xác định dữ liệu (thuộc tính) và hành vi (phương thức) của nó.
Một lớp là sự trừu tượng hóa của một tập các đối tượng	Dữ liệu của các đối tượng khác nhau là khác nhau
	Đối tượng là một thể hiện (instance) của một lớp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

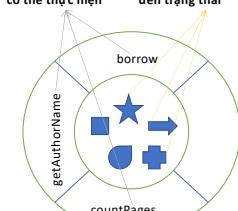
14

14

Lớp vs. Đối tượng (2)

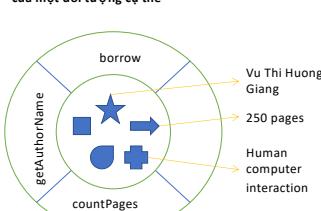
Lớp BOOK

Phương thức: các hành vi đối tượng có thể thực hiện



Đối tượng MyBook

Thuộc tính: các thông tin liên quan đến trạng thái



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Nội dung

1. Trừu trưng hóa
2. Xây dựng lớp
3. Đóng gói và che giấu dữ liệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

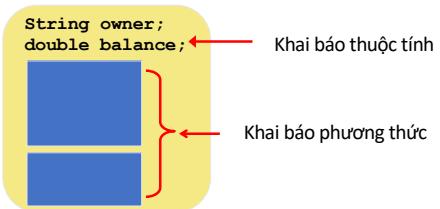
16

16

Thành phần của lớp

❖ Thông tin cần thiết để định nghĩa một lớp

- *Tên (Name)*
 - Tên lớp nên mô tả đối tượng trong thế giới thật
 - Tên lớp nên là số ít, ngắn gọn, và xác định rõ ràng cho sự trừu tượng hóa.
- Danh sách các thuộc tính
- Danh sách các phương thức



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

Cú pháp khai báo lớp

❖ Cú pháp khai báo

```
package tenpackage;  
chi_dinh_truy_cap class TenLop {  
    // Than lop  
}
```

❖ Chỉ định truy cập

- Chỉ định truy cập xác định khả năng nhìn thấy được của một thành phần của chương trình với các thành phần khác của chương trình
- **public:** Lớp có thể được truy cập từ bất cứ đâu, kể cả bên ngoài package chứa lớp đó.
- **Không chỉ định:** Lớp chỉ có thể được truy cập từ bên trong package chứa lớp đó.

```
package oop.cnpm;  
public class Student {  
    ...  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

Package - gói

- ❖ Các lớp được nhóm lại thành package
 - Package bao gồm một tập hợp các lớp có quan hệ logic với nhau
- ❖ Gói (package) giống như thư mục giúp:
 - Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
 - Tránh cho việc đặt tên lớp bị xung đột (trùng tên). Các package khác nhau có thể chứa các lớp có cùng tên
 - Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.
- ❖ Còn được gọi là không gian tên (namespace) trong một số ngôn ngữ lập trình (C++...)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

Package – gói (2)

- ❖ Một số package có sẵn của Java: **java.lang**, **javax.swing**, **java.io...**
- ❖ Package có thể do ta tự đặt
 - Cách nhau bằng dấu ""
 - Quy ước sử dụng ký tự thường để đặt tên package
 - Tên gói phải được viết trên cùng của file mã nguồn
- ❖ Chỉ được phép có 1 câu khai báo gói trong mỗi file mã nguồn, và khai báo này sẽ được áp dụng cho tất cả các dữ liệu trong file đó.
- ❖ Một gói có thể được đặt trong một gói khác
 - Phân cách bằng dấu .
 - Ví dụ **package trungtt.oop;**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

Khai báo Thuộc tính

- ❖ Cú pháp khai báo thuộc tính: Tương tự khai báo biến

```
chi_dinh_truy_cap kieu tenThuocTinh;
```
- ❖ Thuộc tính có thể được khởi tạo khi khai báo. Các giá trị mặc định sẽ được sử dụng nếu không được khởi tạo.
- ❖ Ví dụ

```
access modifier      type
                  |
package com.megabank.models;
public class BankAccount {
    private String owner;
    private double balance = 0.0;
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

Khai báo Phương thức

- ❖ Khai báo: tương tự khai báo hàm
- ❖ Cú pháp

```
chi_dinh_truy_cap kieuTrảVề tênPhươngThức (ds tham số) {
    // Nội dung phương thức
}
```

- ❖ Ví dụ

```
access modifier      return type      method name      parameter list
                  |           |           |
public void debit(double amount) {
    // Method body
    // Java code that implements method behavior
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

Chữ ký phương thức (signature)

- ❖ Mỗi phương thức phải có một chữ ký riêng, phân biệt các phương thức, gồm:
 - Tên phương thức
 - Số lượng các tham số và kiểu của chúng

```
method name      argument type
                  |
public void credit(double amount) {
    ...
}
```

↑
signature



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

Lệnh return

- ❖ Khi phương thức trả về ít nhất một giá trị hoặc một đối tượng thì bắt buộc phải có câu lệnh return để trả điều khiển cho đối tượng gọi phương thức.
- ❖ Nếu phương thức không trả về 1 giá trị nào (void) không cần câu lệnh return
- ❖ Có thể có nhiều lệnh return trong một phương thức; câu lệnh đầu tiên mà chương trình gặp sẽ được thực thi.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

Ví dụ - Khai báo phương thức

```
public boolean checkOdd(int i) {  
    if (i %2 ==0)  
        return true;  
    else  
        return false;  
}  
  
public boolean checkOdd(int i) {  
    return true;  
    return false; //error  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

25

Chỉ định truy cập cho thành viên của lớp

- ❖ **public:** Thuộc tính hoặc phương thức có thể được truy cập từ bất cứ đâu, kể cả bên ngoài lớp, ngoài gói chứa lớp đó.
- ❖ **default/package** (không có chỉ định truy cập): Thuộc tính hoặc phương thức chỉ có thể được truy cập từ bên trong package chứa lớp đó.
- ❖ **private:** Thuộc tính hoặc phương thức chỉ có thể được truy cập trong phạm vi lớp đó
- ❖ **protected:** Thuộc tính hoặc phương thức chỉ có thể được truy cập trong phạm vi lớp đó và từ lớp con kế thừa của lớp đó.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

27

Ví dụ - Khai báo lớp

```
class BankAccount {  
    private String owner;  
    private double balance;  
    public boolean debit(double amount){  
        if (amount > balance)  
            return false;  
        else {  
            balance -= amount;  
            return true;  
        }  
    }  
    public void credit(double amount){  
        balance += amount;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

26

Chỉ định truy cập cho thành viên của lớp

	public	Không có	private
Cùng lớp			
Cùng gói			
Khác gói			



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

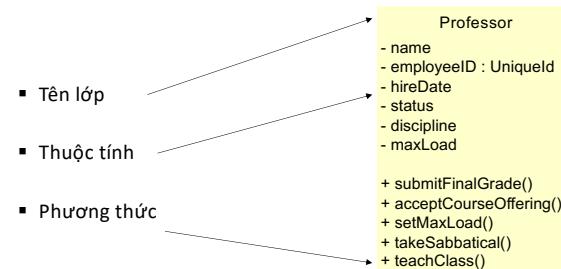
28

Chỉ định truy cập cho thành viên của lớp

	public	Không có	private
Cùng lớp	Yes	Yes	Yes
Cùng gói	Yes	Yes	No
Khác gói	Yes	No	No

Biểu diễn UML (1/3)

- ❖ Lớp (class) được biểu diễn bằng 1 hình chữ nhật với 3 thành phần:



Biểu diễn UML (2/3)

- ❖ Đối tượng: biểu diễn bằng *tên đối tượng:tên lớp*, và các giá trị của thuộc tính.

Student
- name
- address
- studentID
- dateOfBirth

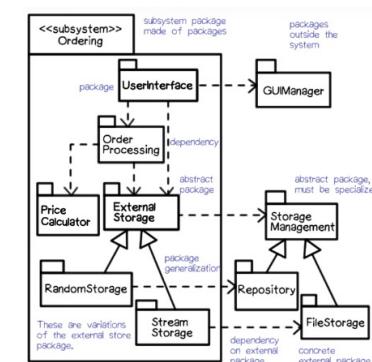
:Student
- name = "M. Modano"
- address = "123 Main St."
- studentID = 9
- dateOfBirth = "03/10/1967"

sv2:Student
- name = "D. Hatcher"
- address = "456 Oak Ln."
- studentID = 2
- dateOfBirth = "12/11/1969"

Class

Biểu diễn UML (3/3)

- ❖ Biểu diễn gói trong UML



Nội dung

1. Trừu tượng hóa
2. Xây dựng lớp
3. Đóng gói và che giấu dữ liệu



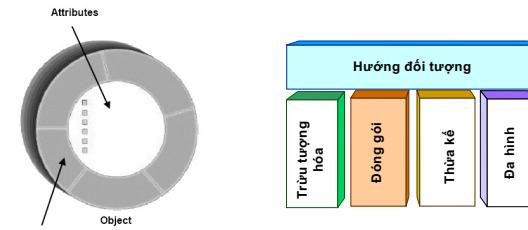
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

33

Đóng gói – Encapsulation (1/3)

- ❖ Là một trong 4 nguyên lý cơ bản của lập trình HDT.
- ❖ Dữ liệu/thuộc tính và hành vi/phương thức được đóng gói trong một lớp.



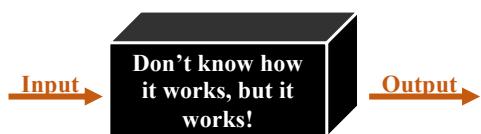
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

34

Đóng gói (2/3)

- ❖ Một đối tượng là một thực thể được đóng gói với mục đích:
 - Cung cấp tập các dịch vụ nhất định
 - Đối tượng được đóng gói có thể được xem như một hộp đen – các công việc bên trong là ẩn so với client
 - Dù thay đổi thiết kế/mã nguồn bên trong nhưng giao diện bên ngoài không bị thay đổi theo



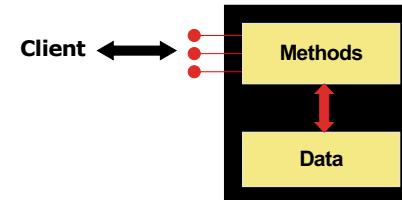
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

35

Đóng gói (3/3)

- ❖ Sau khi đóng gói, một đối tượng có hai khung nhìn:
 - Bên trong: Chi tiết về các thuộc tính và các phương thức của lớp tương ứng với đối tượng
 - Bên ngoài: Các dịch vụ mà một đối tượng có thể cung cấp và cách đối tượng đó tương tác với phần còn lại của hệ thống



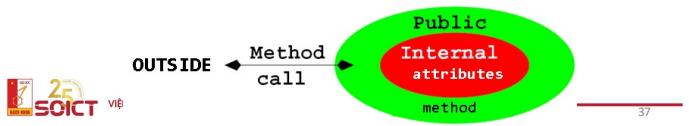
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

36

Che giấu dữ liệu

- ❖ Sử dụng phạm vi truy cập để che giấu dữ liệu: tránh thay đổi trái phép hoặc làm sai lệch dữ liệu
- ❖ Dữ liệu được che giấu ở bên trong lớp bằng cách gán phạm vi truy cập private. Dữ liệu chỉ có thể được truy cập từ các phương thức bên trong lớp
- ❖ Các đối tượng khác muốn truy nhập vào dữ liệu riêng tư này phải thông qua các phương thức của lớp có phạm vi truy cập public.



37

Che giấu dữ liệu (2)

- ❖ Để truy cập và chỉnh sửa các giá trị của dữ liệu, lớp cần phải cung cấp các phương thức
 - Accessor (getter): Trả về giá trị hiện tại của một thuộc tính (dữ liệu)
 - Mutator (setter): Thay đổi giá trị của một thuộc tính
 - Thường là getX và setX, trong đó X là tên thuộc tính



38

Phương thức Get

- ❖ Các phương thức truy vấn Get là các phương thức dùng để hỏi giá trị của các thành viên dữ liệu của một đối tượng
- ❖ Có nhiều loại câu hỏi truy vấn có thể:
 - truy vấn đơn giản ("giá trị của x là bao nhiêu?")
 - truy vấn điều kiện ("thành viên x có lớn hơn 10 không?")
 - truy vấn dẫn xuất ("tổng giá trị của các thành viên x và y là bao nhiêu?")
- ❖ Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng
 - không thay đổi giá trị của thành viên dữ liệu nào.



39

Phương thức Set

- ❖ Các phương thức thiết lập Set là các phương thức dùng để thay đổi giá trị các thành viên dữ liệu
- ❖ Ưu điểm của việc sử dụng các phương thức setter là kiểm soát tính hợp lệ của các thành phần dữ liệu
 - Kiểm tra giá trị đầu vào trước khi gán vào các thuộc tính



40

10

Ví dụ: phương thức get, set

```
class Student{  
    private String name;  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name)  
    {  
        this.name = name;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

Bài tập

- ❖ **Bài 1:** Viết khai báo một gói chứa hai lớp: lớp hình vuông, lớp hình tròn. Viết khai báo lớp hình vuông, lớp hình tròn cùng các thuộc tính thích hợp, các phương thức get/set thích hợp.
- ❖ **Bài 2:** Viết khai báo một lớp Vector gồm 3 thành phần với những phương thức cộng/trừ vector, nhân với 1 hằng số, nhân vô hướng 2 vector.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

43

Ví dụ: phương thức get, set (2)

```
class Student{  
    private String name;  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name)  
    {  
        this.name = name;  
    }  
}
```

```
class Manager{  
    private Student[] students;  
    public initialize()  
    {  
        students = new Student[10];  
        students[0] = new Student();  
        //students[0].name = "Hung"; error  
        students[0].setName("Hung");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

Bài tập

- ❖ **Bài 3:** Viết mã nguồn cho lớp NhanVien như trong hình bên dưới:
 - Lương = Lương cơ bản * Hệ số lương
 - Phương thức inTTin() hiển thị thông tin của đối tượng NhanVien tương ứng.
 - Phương thức tangLuong(double) tăng hệ số lương hiện tại lên một lượng bằng giá trị tham số double truyền vào. Nếu điều này làm cho lương của nhân viên > lương tối đa cho phép thì không cho phép thay đổi, in ra thông báo và trả về false, ngược lại trả về true.
- ❖ Viết các phương thức get và set cho các thuộc tính của lớp NhanVien

```
NhanVien  
-tenNhanVien: String  
-luongCoBan: double  
-heSoLuong: double  
+LUONG_MAX: double  
+tangLuong (double) :boolean  
+tinhLuong () : double  
+inTTin ()
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

44



Bài 4: Khởi tạo và sử dụng đối tượng

1

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



2

Bài giảng E-learning

- ❖ Phương thức khởi tạo, khai báo và sử dụng đối tượng
 - https://www.youtube.com/watch?v=rw_bpkesNH0
 - <https://www.youtube.com/watch?v=MTCgdBLrlw>
 - <https://www.youtube.com/watch?v=XznNdY3Bfvg>
- ❖ Quản lý bộ nhớ: Stack và Heap
 - <https://www.youtube.com/watch?v=450maTzSlvA>
 - <https://www.youtube.com/watch?v=1rLHJqx98Q>
- ❖ Equals và ==
 - <https://www.youtube.com/watch?v=qQe69w1YF54>
- ❖ Java finalize method
 - <https://www.youtube.com/watch?v=j3fRK7T1pQo>



3

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



4

1. Phương thức khởi tạo

- Dữ liệu cần được khởi tạo trước khi sử dụng
 - Lỗi khởi tạo là một trong các lỗi phổ biến
- Với kiểu dữ liệu đơn giản, sử dụng toán tử =
- Với đối tượng → Cần dùng phương thức khởi tạo



5

5

1. Phương thức khởi tạo

- Là phương thức đặc biệt được gọi tự động khi tạo ra đối tượng
- Mục đích chính: Khởi tạo cho các thuộc tính của đối tượng



7

7

Khởi tạo và hủy bỏ đối tượng

- Mỗi đối tượng khi tồn tại và hoạt động được hệ điều hành cấp phát một vùng nhớ để lưu lại các giá trị của dữ liệu thành phần
 - Phải được thực hiện tự động trước khi người lập trình có thể tác động lên đối tượng
- Khi tạo ra đối tượng HĐH sẽ gán giá trị khởi tạo cho các dữ liệu thành phần
 - Sử dụng hàm/phương thức khởi tạo
- Ngược lại khi kết thúc cần phải giải phóng hợp lý tất cả các bộ nhớ đã cấp phát cho đối tượng.
 - Java: JVM
 - C++: Hàm hủy (destructor)



6

6

1. Phương thức khởi tạo

- Mỗi lớp phải chứa ít nhất một constructor
 - Có nhiệm vụ tạo ra một thể hiện mới của lớp
 - Tên của constructor trùng với tên của lớp
 - Constructor không có kiểu dữ liệu trả về
- Ví dụ:

```
public BankAccount(String o, double b){  
    owner = o;  
    balance = b;  
}
```



8

8

1. Phương thức khởi tạo

- ❖ Phương thức khởi tạo **có thể dùng** các chỉ định truy cập
 - **public**
 - **private**
 - Không có (mặc định – phạm vi package)
- ❖ Một phương thức khởi tạo **không thể dùng** các từ khóa **abstract, static, final, native, synchronized**.
- ❖ Các phương thức khởi tạo không được xem như là **thành viên của lớp**.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

2. Các loại phương thức khởi tạo

- ❖ 2 loại phương thức khởi tạo
 - Phương thức khởi tạo mặc định (Phương thức khởi tạo không tham số)
 - Phương thức khởi tạo có tham số



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

Phương khởi tạo mặc định (default constructor)

- ❖ Là phương thức khởi tạo **KHÔNG THAM SỐ**

```
public BankAccount(){  
    owner = "noname"; balance = 100000;  
}
```

- ❖ Một lớp nên có phương thức khởi tạo mặc định



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

12

Phương thức khởi tạo mặc định

- ❖ Khi LTV không viết một phương khởi tạo nào trong lớp
 - JVM cung cấp phương thức khởi tạo mặc định
 - Phương thức khởi tạo mặc định do JVM cung cấp có chỉ định truy cập giống như lớp của nó

```
public class MyClass{  
    public static void main(String args){  
        //...  
    }  
  
    public class MyClass{  
        public MyClass(){  
        }  
        public static void main(String args){  
            //...  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

Phương thức khởi tạo có tham số

- ❖ Một phương thức khởi dựng có thể có các tham số truyền vào
- ❖ Dùng khi muốn khởi tạo giá trị cho các thuộc tính
- ❖ Ví dụ:

```
public BankAccount(String o, double b){  
    owner = o;  
    balance = b;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

3. Khai báo và khởi tạo đối tượng

- ❖ Đối tượng được tạo ra, thể hiện hóa (instantiate) từ một mẫu chung (lớp).
- ❖ Các đối tượng phải được khai báo *kiểu* của đối tượng trước khi sử dụng:
 - Kiểu của đối tượng là lớp các đối tượng
 - Ví dụ:
 - String strName;
 - BankAccount acc;



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

15

4

3. Khai báo và khởi tạo đối tượng

- ❖ Đối tượng cần được khởi tạo trước khi sử dụng
 - Sử dụng toán tử = để gán
 - Sử dụng từ khóa **new** với constructor để khởi tạo đối tượng:
 - Từ khóa **new** dùng để tạo ra một đối tượng mới
 - Tự động gọi phương thức khởi tạo tương ứng
 - Một đối tượng được khởi tạo mặc định là **null**
- ❖ Đối tượng được thao tác thông qua *tham chiếu* (~ con trỏ).
- ❖ Ví dụ:

```
BankAccount acc1;  
acc1 = new BankAccount();
```

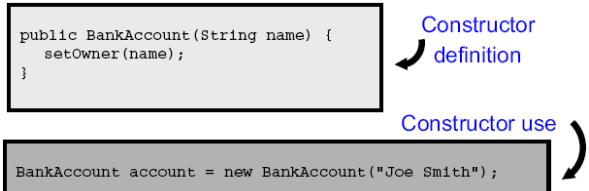


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

3. Khai báo và khởi tạo đối tượng

- ❖ Phương thức khởi tạo **không có giá trị trả về**, nhưng khi sử dụng với từ khóa **new** trả về một tham chiếu đến đối tượng mới



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

3. Khai báo và khởi tạo đối tượng

- ❖ Có thể kết hợp vừa khai báo và khởi tạo đối tượng
- ❖ Cú pháp:

```
Ten_lop ten_doi_tuong = new  
Pthuc_khoi_tao(ds_tham_so);
```

- ❖ Ví dụ:

```
BankAccount account = new  
BankAccount();
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

3. Khai báo và khởi tạo đối tượng

- ❖ Mảng các đối tượng được khai báo giống như mảng dữ liệu cơ bản
- ❖ Mảng các đối tượng được khởi tạo mặc định với giá trị **null**.

- ❖ Ví dụ:

```
Employee emp1 = new Employee(123456);  
Employee emp2;  
emp2 = emp1;
```



```
Department dept[] = new Department[100];  
Test[] t = {new Test(1), new Test(2)};
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

Ví dụ 1

```
class BankAccount{  
    private String owner;  
    private double balance;  
}  
  
public class Test{  
    public static void main(String args[]){  
        BankAccount acc1 = new BankAccount();  
    }  
}
```

→ Phương thức khởi tạo mặc định do Java cung cấp.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

Ví dụ 3

```
public class BankAccount {  
    private String owner;  
    private double balance;  
    public BankAccount(String name) {  
        setOwner(name);  
    }  
    public void setOwner(String o) {  
        owner = o;  
    }  
}  
  
public class Test {  
    public static void main(String args[]){  
        BankAccount account1 = new BankAccount(); //Error  
        BankAccount account2 = new BankAccount("Hoang");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

Ví dụ 2

```
public class BackAccount{  
    private String owner;  
    private double balance;  
    public BankAccount(){  
        owner = "noname";  
    }  
}  
  
public class Test{  
    public static void main(String args[]){  
        BankAccount acc1 = new BankAccount();  
    }  
}
```

→ Phương thức khởi tạo mặc định tự viết.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



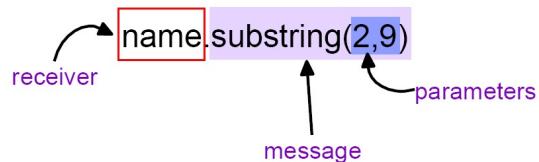
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

24

4. Sử dụng đối tượng

- ❖ Đối tượng cung cấp các hoạt động phức tạp hơn các kiểu dữ liệu nguyên thủy
- ❖ Đối tượng đáp ứng lại các thông điệp
 - Toán tử `".` được sử dụng để gửi một thông điệp đến một đối tượng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

25

4. Sử dụng đối tượng (2)

- ❖ Để gọi thành viên (dữ liệu hoặc thuộc tính) của lớp hoặc đối tượng, sử dụng toán tử `"."`
- ❖ Nếu gọi phương thức ngay trong lớp thì toán tử `"."` không cần thiết.

```
BankAccount account = new BankAccount();
account.setOwner("Smith");
account.credit(1000.0);
System.out.println(account.getBalance());
...
```

```
BankAccount method
public void credit(double amount) {
    setBalance(getBalance() + amount);
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

26

```
public class BankAccount{
    private String owner;
    private double balance;
    public BankAccount(String name) {
        setOwner(name);
        // Là viết tắt của this.setOwner(name)
    }
    public void setOwner(String o){ owner = o; }
    public String getOwner(){ return owner; }
}
public class Test{
    public static void main(String args[]){
        BankAccount acc1 = new BankAccount("");
        BankAccount acc2 = new BankAccount("Hong");
        acc1.setOwner("Hoa");
        System.out.println(acc1.getOwner()
            + "+" + acc2.getOwner());
    }
}
```

27

27

Tự tham chiếu – this

- ❖ Cho phép truy cập vào đối tượng hiện tại của lớp.
- ❖ Quan trọng khi hàm/phương thức thành phần thao tác trên hai hay nhiều đối tượng.
- ❖ Xóa đi sự nhập nhằng giữa một biến cục bộ, tham số với thành phần dữ liệu của lớp
- ❖ Không dùng bên trong các khối lệnh static



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

28

```

public class BankAccount{
    private String owner;
    private double balance;
    public BankAccount() { }
    public void setOwner(String owner){
        this.owner = owner;
    }
    public String getOwner(){ return owner; }
}
public class Test{
    public static void main(String args[]){
        BankAccount acc1 = new BankAccount();
        BankAccount acc2 = new BankAccount();
        acc1.setOwner("Hoa");
        acc2.setOwner("Hong");
        System.out.println(acc1.getOwner() + " " +
                           acc2.getOwner());
    }
}

```

29

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
- 5. Quản lý bộ nhớ và so sánh đối tượng**
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập

30

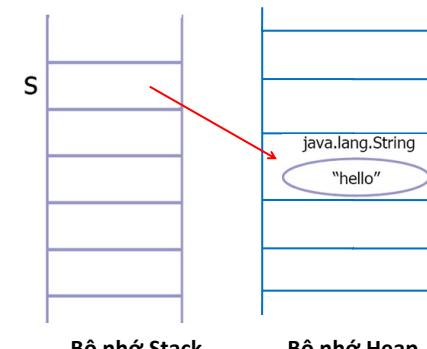
5. Quản lý bộ nhớ và so sánh đối tượng

- ❖ Java không sử dụng con trỏ nên các địa chỉ bộ nhớ không thể bị ghi đè lên một cách ngẫu nhiên hoặc cố ý.
- ❖ Các vấn đề định vị và tái định vị bộ nhớ, quản lý bộ nhớ do JVM kiểm soát, hoàn toàn trong suốt với lập trình viên.
- ❖ Lập trình viên không cần quan tâm đến việc ghi dấu các phần bộ nhớ đã cấp phát trong heap để giải phóng sau này.

Bộ nhớ Heap

```
String s = new String("hello");
```

- s là biến tham chiếu, lưu trên Stack
- Giá trị của s là địa chỉ của vùng nhớ Heap lưu trữ đối tượng s tham chiếu tới
- Bộ nhớ Heap sử dụng để ghi thông tin được tạo bởi toán tử `new`.

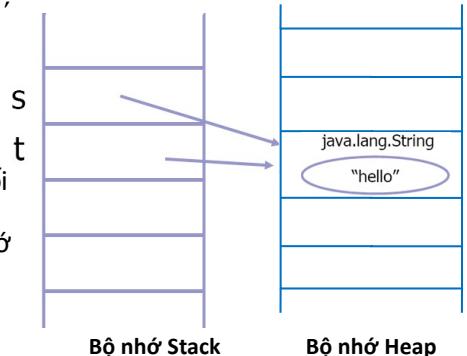


32

Bộ nhớ Heap

```
String s = new String("hello");  
String t = s;
```

- s và t cùng tham chiếu tới một đối tượng (cùng trỏ tới một vùng nhớ trên heap)



So sánh đối tượng

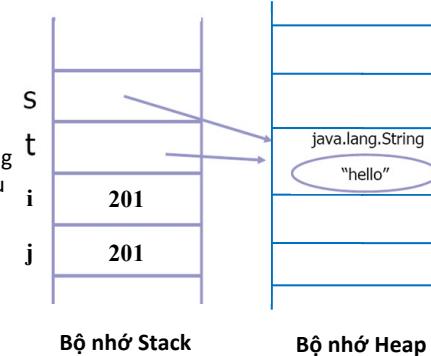
- Đối với các kiểu dữ liệu nguyên thủy, toán tử == kiểm tra xem chúng có giá trị bằng nhau hay không
- Ví dụ:

```
int a = 1;  
int b = 1;  
if (a==b) ... // true
```

Bộ nhớ Stack

```
String s = new String("hello");  
String t = s;  
int i = 201;  
int j = i;
```

- Giá trị cục bộ trong bộ nhớ Stack được sử dụng như con trỏ tham chiếu tới Heap
- Giá trị của dữ liệu nguyên thủy được ghi trực tiếp trong Stack



So sánh đối tượng (2)

- Đối với các đối tượng, toán tử == kiểm tra xem hai đối tượng có đồng nhất hay không, (cùng tham chiếu đến một đối tượng hay không)
- Ví dụ:

a và b tham chiếu
tới 2 đối tượng
khác nhau

```
Employee a = new Employee(1);  
Employee b = new Employee(1);  
if (a==b) ... // false
```

a và b cùng
tham chiếu tới 1
đối tượng

```
Employee a = new Employee(1);  
Employee b = a;  
if (a==b) ... // true
```

So sánh đối tượng (3)

❖ Phương thức equals

- Đối với kiểu dữ liệu nguyên thủy: Không tồn tại.
- Đối với các đối tượng: Bất kỳ đối tượng nào cũng có phương thức này, dùng để so sánh giá trị của đối tượng
- Phương thức equals kế thừa từ lớp Object (chi tiết xem bài [Kết tập và kế thừa](#))
- Cài đặt mặc định của phương thức equals là như toán tử ==. Cần cài đặt lại để so sánh 2 đối tượng dựa trên từng thuộc tính



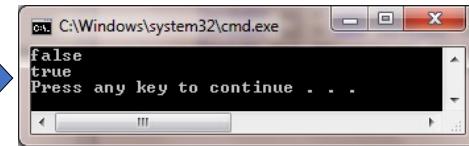
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

Ví dụ == và equals – Lớp Integer

```
public class Equivalence {  
    public static void main(String[] args) {  
        Integer n1 = new Integer(47);  
        Integer n2 = new Integer(47);  
        System.out.println(n1 == n2);  
        System.out.println(n1.equals(n2));  
    }  
}
```



Lớp Integer (lớp cung cấp trong Java SDK) đã cài đặt lại phương thức equals của lớp Object, nên n1.equals(n2) trả về true



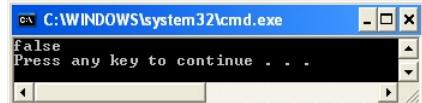
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

Ví dụ sử dụng equals với lớp tự viết

```
class Value {  
    int i;  
}  
  
public class EqualsMethod2 {  
    public static void main(String[] args) {  
        Value v1 = new Value();  
        Value v2 = new Value();  
        v1.i = v2.i = 100;  
        System.out.println(v1.equals(v2));  
    }  
}
```



Lớp Value (LTV tự viết) chưa cài đặt lại phương thức equals của lớp Object, nên v1.equals(v2) trả về false, giống như toán tử ==



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10

6. Hủy bỏ đối tượng

❖ Trong C#, C++:

- Sử dụng phương thức hủy (destructor)
- Phương thức hủy là phương thức tự động được gọi trước khi đối tượng được hủy
- Phương thức hủy thường dùng để dọn dẹp bộ nhớ, thu hồi tài nguyên (VD đối tượng khi hoạt động cần truy cập tới file/CSDL, cấp phát bộ nhớ động)

❖ Trong Java:

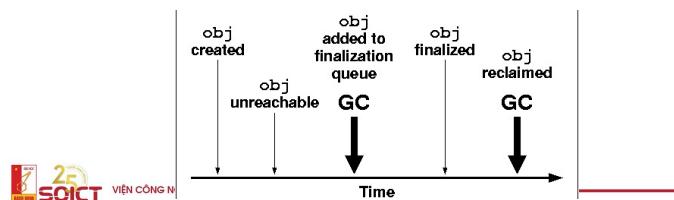
- Không có khái niệm phương thức hủy
- Sử dụng phương thức finalize()

Bộ thu gom rác (Garbage Collector)

- ❖ Một tiến trình chạy ngầm gọi đến bộ “thu gom rác” để phục hồi lại phần bộ nhớ mà các đối tượng không tham chiếu đến (tái định vị)
- ❖ Các đối tượng không có tham chiếu đến được gán null.
- ❖ Bộ thu gom rác định kỳ quét qua danh sách các đối tượng của JVM và phục hồi các tài nguyên của các đối tượng không có tham chiếu.

Phương thức void finalize()

- ❖ Lớp nào cũng có phương thức finalize() – được thực thi ngay lập tức khi quá trình thu gom xảy ra
- ❖ Thường chỉ sử dụng cho các trường hợp đặc biệt để “tự dọn dẹp” các tài nguyên sử dụng khi đối tượng được gc giải phóng
 - Ví dụ cần đóng các socket, file,... nên được xử lý trong luồng chính trước khi các đối tượng bị ngắt bỏ tham chiếu.
- ❖ Có thể coi là phương thức hủy (destructor) của lớp mặc dù Java không có khái niệm này.



Bộ thu gom rác (2)

- ❖ JVM quyết định khi nào thực hiện thu gom rác:
 - Thông thường sẽ thực thi khi thiếu bộ nhớ
 - Tại thời điểm không dự đoán trước
- ❖ Không thể ngăn quá trình thực hiện của bộ thu gom rác nhưng có thể yêu cầu thực hiện sớm hơn:
`System.gc(); hoặc Runtime.gc();`

Nội dung

1. Phương thức khởi tạo
2. Các loại phương thức khởi tạo
3. Khai báo và khởi tạo đối tượng
4. Sử dụng đối tượng
5. Quản lý bộ nhớ và so sánh đối tượng
6. Hủy bỏ đối tượng
7. **Ví dụ và bài tập**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

45

Student.java

```
package example;
public class Student {
    private int year;
    private String name;

    public Student(int year, String name) {
        this.year = year;
        this.name = name;
    }

    public int getYear() {
        return year;
    }

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



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

47

Bài tập 1

❖ Viết lớp Student

- name
- year
- 1 phương thức khởi tạo
 - Student(String name, int year)
- Tự tạo phương thức getter, setter cho đủ dùng
- Đảm bảo đóng gói, che dấu dữ liệu

❖ Lớp Test

- Nhập số phần tử cho mảng Student (trong 1 lớp học)
- Nhập lần lượt các Student
- In ra danh sách tên Student trong lớp và hiển thị tổng số tuổi của các Student



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

46

Test.java

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Nhập số SV:");
        int N = scanner.nextInt();
        Student[] cls = new Student[N];
        for (int i = 0; i < N; ++i) {
            System.out.print("Nhập SV thứ " + (i + 1));
            System.out.print("Name: ");
            String name = scanner.next();
            System.out.print("Year: ");
            int year = scanner.nextInt();
            cls[i] = new Student(year, name);
        }
        scanner.close();

        int total = 0;
        System.out.print("Danh sách lớp: ");
        for (int i = 0; i < N; ++i) {
            total += 2012 - cls[i].getYear();
            System.out.print(cls[i].getName());
        }
        System.out.print("Tổng số tuổi: " + total);
    }
}
```

48

12

Bài tập 2

- ❖ Viết mã nguồn cho lớp NhanVien
(đã làm trong bài học trước)
- ❖ Viết phương thức khởi tạo với các tham số cần thiết để khởi tạo cho các thuộc tính của lớp NhanVien.
- Viết lớp TestNV trong đó tạo ra 2 đối tượng của lớp NhanVien, thực hiện truyền thông điệp đến các đối tượng vừa tạo để hiển thị thông tin, hiển thị lương, tăng lương...

NhanVien
-tenNhanVien: String
-luongCoBan: double
-heSoLuong: double
+tangLuong (double) :boolean
+tinhLuong () : double
+inTTin ()

Bài 5: Các kỹ thuật xây dựng lớp và sử dụng đối tượng

1

Nội dung

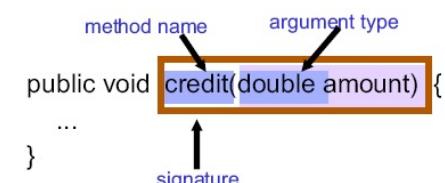
1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Truyền tham số cho phương thức
4. Một số lớp tiện ích trong Java
5. Ví dụ và bài tập

Nội dung

1. **Chồng phương thức**
2. Thành viên ĐT và thành viên lớp
3. Truyền tham số cho phương thức
4. Một số lớp tiện ích trong Java
5. Ví dụ và bài tập

Nhắc lại về phương thức

- ❖ Mỗi phương thức phải có một chữ ký riêng
- ❖ Chữ ký của phương thức bao gồm:
 - Tên phương thức
 - Số lượng các đối số và kiểu của chúng



3

4

1.1. Chồng phương thức

- ❖ Chồng phương thức (Method Overloading): Các phương thức trong cùng một lớp có thể trùng tên nhưng chữ ký phải khác nhau:
 - Số lượng tham số khác nhau
 - Nếu cùng số lượng tham số thì kiểu dữ liệu các tham số phải khác nhau
- ❖ Mục đích:
 - Tên trùng nhau để mô tả bản chất công việc
 - Thuận tiện cho lập trình vì không cần phải nhớ quá nhiều tên phương thức mà chỉ cần nhớ một tên và lựa chọn các tham số cho phù hợp.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1.1. Chồng phương thức (3)

- ❖ Ví dụ 2:

```
class MyDate {  
    int year, month, day;  
    public boolean setMonth(int m) { ...}  
    public boolean setMonth(String s) { ...}  
}  
  
public class Test{  
    public static void main(String args[]){  
        MyDate d = new MyDate();  
        d.setMonth(9);  
        d.setMonth("September");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

1.1. Chồng phương thức (2)

- ❖ Ví dụ 1:
 - Phương thức println() trong System.out.println() có 10 khai báo với các tham số khác nhau: boolean, char[], char, double, float, int, long, Object, String, và một không có tham số.
 - Không cần sử dụng các tên khác nhau (chẳng hạn "printString" hoặc "printDouble") cho mỗi kiểu dữ liệu muốn hiển thị.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Một số chú ý với chồng phương thức

- ❖ Các phương thức chỉ được xem xét là chồng khi chúng thuộc cùng một lớp
- ❖ Chỉ nên sử dụng kỹ thuật này với các phương thức có cùng mục đích, chức năng; tránh lạm dụng
- ❖ Khi dịch, trình dịch căn cứ vào số lượng hoặc kiểu dữ liệu của tham số để quyết định gọi phương thức nào phù hợp.
 - Nếu không chọn được hoặc chọn được nhiều hơn 1 phương thức thì sẽ báo lỗi.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

7

2

Thảo luận

- Cho phương thức sau đây:

```
public double test(String a, int b)
```

- Hãy chọn ra các phương thức chồng cho phương thức trên:
 - void test(String b, int a)
 - public double test(String a)
 - private int test(int b, String a)
 - private int test(String a, int b)
 - double test(double a, int b)
 - double test(int b)
 - public double test(String a, long b)

Thảo luận

```
void prt(String s) { System.out.println(s); }  
void f2(short x) { prt("f3(short)"); }  
void f2(int x) { prt("f3(int)"); }  
void f2(long x) { prt("f5(long)"); }  
void f2(float x) { prt("f5(float)"); }
```

- Điều gì xảy ra nếu thực hiện:

- f2(5);
- char x='a'; f2(x);
- byte y=0; f2(y);
- float z = 0; f2(z);

f3<int>
f3<int>
f3<short>
f5<float>

- Điều gì xảy ra nếu gọi f2(5.5)?

Error: cannot find symbol: method f2(double)

Thảo luận

```
void prt(String s) { System.out.println(s); }  
void f1(char x) { prt("f1(char)"); }  
void f1(byte x) { prt("f1(byte)"); }  
void f1(short x) { prt("f1(short)"); }  
void f1(int x) { prt("f1(int)"); }  
void f1(long x) { prt("f1(long)"); }  
void f1(float x) { prt("f1(float)"); }  
void f1(double x) { prt("f1(double)"); }
```

- Điều gì xảy ra nếu thực hiện:

- f1(5);
- char x='a'; f1(x);
- byte y=0; f1(y);
- float z = 0; f1(z);...

Ví dụ

```
class MyClass {  
    public void myMethod(int a, long b){  
    }  
    public void myMethod(long a, int b) { // overloading  
    }  
}  
  
public class Test {  
    public static void main(String args[]){  
        MyClass m = new MyClass();  
        m.myMethod(); // error do không có method phù hợp  
        m.myMethod(9, 10); // error do có 2 phiên bản method phù hợp  
    }  
}
```

1.2. Chồng phương thức khởi tạo

- ❖ Trong nhiều tình huống khác nhau cần khởi tạo đối tượng theo nhiều cách khác nhau
- ❖ → Cần xây dựng các phương thức khởi tạo khác nhau cho đối tượng theo nguyên lý chồng phương thức (constructor overloading).



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

1.3. Từ khóa this

- ❖ Nhắc lại: Tự tham chiếu đến đối tượng hiện tại, sử dụng bên trong lớp tương ứng với đối tượng muốn tham chiếu.
- ❖ Sử dụng thuộc tính hoặc phương thức của đối tượng thông qua toán tử "", ví dụ:

```
public class BankAccount{  
    private String owner;  
    public void setOwner(String owner){  
        this.owner = owner;  
    }  
    public BankAccount() { this.setOwner("noname"); }  
    ...  
}  
❖ Gọi đến phương thức khởi tạo khác của lớp:  
    • this(danh_sach_tham_so); //neu co tham so
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Ví dụ

```
public class BankAccount{  
    private String owner;  
    private double balance;  
    public BankAccount(){owner = "noname";}  
    public BankAccount(String o, double b){  
        owner = o; balance = b;  
    }  
}  
public class Test{  
    public static void main(String args[]){  
        BankAccount acc1 = new BankAccount();  
        BankAccount acc2 = new BankAccount("Thuy", 100);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Ví dụ

```
public class Ship {  
    private double x=0.0, y=0.0  
    private double speed=1.0, direction=0.0;  
    public String name;  
  
    public Ship(String name) {  
        this.name = name;  
    }  
    public Ship(String name, double x, double y) {  
        this(name); this.x = x; this.y = y;  
    }  
    public Ship(String name, double x, double y, double  
    speed, double direction) {  
        this(name, x, y);  
        this.speed = speed;  
        this.direction = direction;  
    }  
    //continue...
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

```

//(cont.)
private double degreesToRadian(double degrees) {
    return(degrees * Math.PI / 180.0);
}
public void move() {
    move(1);
}
public void move(int steps) {
    double angle = degreesToRadians(direction);
    x = x + (double)steps*speed*Math.cos(angle);
    y = y + (double)steps*speed*Math.sin(angle);
}
public void printLocation() {
    System.out.println(name + " is at (" +
        + x + "," + y + ").");
}
} //end of Ship class

```

17

17

2.1. Thành viên static

❖ Trong Java

- Các thành viên bình thường là thành viên thuộc về đối tượng
- Thành viên thuộc về lớp được khai báo là static

❖ Cú pháp khai báo thành viên static:

- chi_dinh_truy_cap static kieu_du_lieu tenBien;

❖ Ví dụ:

```

public class MyDate {
    public static long getMillisSinceEpoch() {
        ...
    }
    ...
    long millis = MyDate.getMillisSinceEpoch();
}

```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Truyền tham số cho phương thức
4. Một số lớp tiện ích trong Java
5. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

19

Ví dụ lớp JOptionPane trong javax.swing

❖ Thuộc tính

Field Summary	
static int CANCEL_OPTION	Return value from class method if CANCEL is chosen
static int CLOSED_OPTION	Return value from class method if user closes window
static int CANCEL_OPTION or NO_OPTION	CANCEL_OPTION or NO_OPTION.
static int DEFAULT_OPTION	Type used for showConfirmDialog.
static int ERROR_MESSAGE	Used for error messages.
static int WARNING_MESSAGE	Used for warning messages.
static int YES_NO_CANCEL_OPTION	Type used for showConfirmDialog.
static int YES_NO_OPTION	Type used for showConfirmDialog.
static int YES_OPTION	Return value from class method if YES is chosen.

❖ Phương thức:

static void showMessageDialog(Component parentComponent, Object message)	Brings up an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)	Brings up a dialog displaying a message specifying all parameters



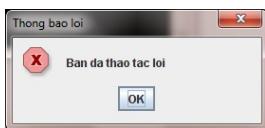
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

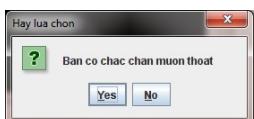
5

Ví dụ - sử dụng thuộc tính và phương thức static lớp JOptionPane

```
JOptionPane.showMessageDialog(null,"Ban da thao  
tac loi", "Thong bao loi",  
JOptionPane.ERROR_MESSAGE);
```



```
JOptionPane.showConfirmDialog(null,"Ban co chac  
chan muon thoat?", "Hay lua chon",  
JOptionPane.YES_NO_OPTION);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

2.1. Thành viên static (2)

- ❖ Thay đổi giá trị của một thành viên **static** trong một đối tượng của lớp sẽ thay đổi giá trị của thành viên này của tất cả các đối tượng khác của lớp đó.
- ❖ Các phương thức **static** chỉ có thể truy cập vào các thuộc tính **static** và chỉ có thể gọi các phương thức **static** trong cùng lớp.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

Ví dụ - sử dụng thuộc tính và phương thức static lớp JOptionPane (2)

- ❖ Object[] options = { "OK", "CANCEL" };
- ❖ JOptionPane.showOptionDialog(null,"Nhan OK de tiep tục", "Canh bao", JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,null,options,options[0]);



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

Ví dụ 1

```
class TestStatic{  
    public static int iStatic;  
    public int iNonStatic;  
}  
public class TestS {  
    public static void main(String[] args) {  
        TestStatic obj1 = new TestStatic();  
  
        obj1.iStatic = 10; obj1.iNonStatic = 11;  
        System.out.println(obj1.iStatic + "," + obj1.iNonStatic);  
  
        TestStatic obj2 = new TestStatic();  
        System.out.println(obj2.iStatic + "," + obj2.iNonStatic);  
  
        obj2.iStatic = 12;  
        System.out.println(obj1.iStatic + "," + obj1.iNonStatic);  
    }  
}
```

```
10,11  
10,0  
12,11
```

25

Ví dụ 2

```
public class Demo {  
    int i = 0;  
    void tang(){ i++; }  
    public static void main(String[] args) {  
        tang();  
        System.out.println("Gia tri cua i la" +  
            i);  
    }  
}
```

non-static method tang() cannot be referenced from a static context
non-static variable i cannot be referenced from a static context



26

26

2.2. Thành viên hằng (2)

- Thông thường các hằng số liên quan đến lớp được khai báo là **static final** nhằm giúp truy cập dễ dàng

```
public class MyDate {  
    public static final long SECONDS_PER_YEAR =  
        31536000;  
    ...  
    long years = MyDate.currentTimeMillis() /  
        (1000*MyDate.SECONDS_PER_YEAR);
```

javax.swing
Class JOptionPane

ERROR_MESSAGE

```
public static final int ERROR_MESSAGE
```



28

28

2.2. Thành viên hằng

- Một thuộc tính/phương thức không thể thay đổi giá trị/nội dung trong quá trình sử dụng.

- Cú pháp khai báo:

```
chi_dinh_truy_cap final kieu_du_lieu  
TEN_HANG = gia_tri;
```

- Ví dụ:

```
final double PI = 3.141592653589793;  
public final int VAL_THREE = 39;  
private final int[] A = { 1, 2, 3, 4, 5, 6 };
```



27

27

Instance member vs. Class member

Thành viên đối tượng

- Thuộc tính/phương thức chỉ được truy cập thông qua **đối tượng**
- Mỗi đối tượng có 1 bản sao riêng của 1 thuộc tính đối tượng
- Giá trị của 1 thuộc tính đối tượng của các đối tượng khác nhau là khác nhau.

Thành viên lớp

- Thuộc tính/phương thức có thể được truy cập thông qua **lớp**
- Các đối tượng có chung 1 bản sao của 1 thuộc tính lớp
- Giá trị của 1 thuộc tính lớp của các đối tượng khác nhau là giống nhau.



29

7

Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. **Truyền tham số cho phương thức**
4. Một số lớp tiện ích trong Java
5. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

30

3. Truyền tham số cho phương thức (2)

- ❖ Java truyền mọi tham số cho phương thức dưới dạng giá trị (pass-by-value): Truyền giá trị/bản sao của tham số thực
 - Với tham số có kiểu dữ liệu tham trị (kiểu dữ liệu nguyên thủy): Truyền giá trị/bản sao của các biến nguyên thủy truyền vào
 - Với tham số có kiểu dữ liệu tham chiếu (mảng và đối tượng): Truyền giá trị/bản sao của tham chiếu gốc truyền vào

→ Thay đổi tham số hình thức không làm ảnh hưởng đến tham số thực



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

32

3. Truyền tham số cho phương thức

- ❖ Có thể sử dụng bất kỳ kiểu dữ liệu nào cho tham số của phương thức hoặc constructor
 - Kiểu dữ liệu nguyên thủy
 - Kiểu dữ liệu tham chiếu: mảng và đối tượng
- ❖ Ví dụ

```
public Polygon polygonFrom(Point[] corners) {  
    // method body goes here  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

31

3.1. Với kiểu dữ liệu tham trị

- ❖ Các giá trị nguyên thủy không thể thay đổi khi truyền như một tham số
- ❖ Phương thức swap này có hoạt động đúng không?

```
public void swap(int var1, int var2) {  
    int temp = var1;  
    var1 = var2;  
    var2 = temp;  
}
```



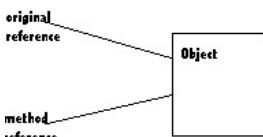
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

33

3.2. Với kiểu dữ liệu tham chiếu

- Thực ra là truyền bản sao của tham chiếu gốc, chứ không phải truyền tham chiếu gốc hoặc truyền đối tượng (pass the references by value, not the original reference or the object)



- Sau khi truyền cho phương thức, đối tượng có ít nhất 2 tham chiếu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

3.2. Với kiểu dữ liệu tham chiếu-ví dụ 1

```
public class Student {  
    private int year;  
    private String name;  
  
    public Student(String name, int year) {  
        this.year = year;  
        this.name = name;  
    }  
  
    public int getYear() {  
        return year;  
    }  
    public void setYear(int year) {  
        this.year = year;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

3.2. Với kiểu dữ liệu tham chiếu-ví dụ 1

```
public class Test {  
    public static void change(Student std){  
        std.setYear(2000);  
    }  
    public static void main(String[] args) {  
        Student std = new Student("Nam", 1990);  
        System.out.println(std.getYear());  
        change(std);  
        System.out.println(std.getYear());  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

4.2. Với kiểu dữ liệu tham chiếu-ví dụ 2

```
public class Test {  
    public static void change(Student std){  
        std = new Student("Hung", 1995);  
    }  
    public static void main(String[] args) {  
        Student std = new Student("Nam", 1990);  
        System.out.println(std.getYear());  
        change(std);  
        System.out.println(std.getYear());  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

Ví dụ

```
public class Point {  
    private double x;  
    private double y;  
    public Point() {}  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
    public void setX(double x) {  
        this.x = x;  
    }  
    public void setY(double y) {  
        this.y = y;  
    }  
    public void printPoint() {  
        System.out.println("X: " + x + " Y: " + y);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

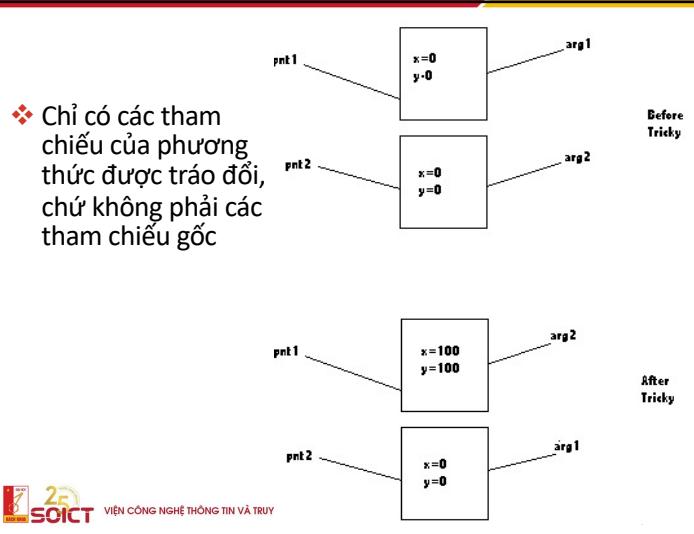
38

```
public class Test {  
    public static void tricky(Point arg1, Point arg2) {  
        arg1.setX(100); arg1.setY(100);  
        Point temp = arg1;  
        arg1 = arg2; arg2 = temp;  
    }  
    public static void main(String[] args) {  
        Point pnt1 = new Point(0,0);  
        Point pnt2 = new Point(0,0);  
        pnt1.printPoint(); pnt2.printPoint();  
        System.out.println(); tricky(pnt1, pnt2);  
        pnt1.printPoint(); pnt2.printPoint();  
    }  
}
```

```
X: 0.0 Y: 0.0  
X: 0.0 Y: 0.0  
X: 100.0 Y: 100.0  
X: 0.0 Y: 0.0  
Press any key to continue . . .
```

39

39



40

3.3. Truyền số lượng tham số tùy ý

❖ Được gọi là varargs. Cú pháp:

- ten_phuong_thuc(Kieu_dl... ten_tham_so)

❖ Ví dụ 1:

- Khai báo:

```
public PrintStream printf(String format,  
                         Object... args)
```
- Sử dụng
 - ```
System.out.printf ("%s: %d, %s\n",
 name, idnum, address);
```
  - ```
System.out.printf ("%s: %d, %s, %s, %s\n",  
                   name, idnum, address, phone, email);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

10

❖ Ví dụ 2

```
public Polygon polygonFrom(Point... corners) {  
    int numberofSides = corners.length;  
    double squareOfSide1, lengthOfSide1;  
    squareOfSide1 = (corners[1].x - corners[0].x)  
        *(corners[1].x - corners[0].x)  
        + (corners[1].y - corners[0].y)  
        *(corners[1].y - corners[0].y) ;  
    lengthOfSide1 = Math.sqrt(squareOfSide1);  
    //create & return a polygon connecting the Points  
}
```

❖ Nhận xét

- **corners** được coi như một mảng
- Phương thức có thể được gọi bằng cách truyền một mảng hoặc một loạt các tham số truyền vào

42

42

Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Truyền tham số cho phương thức
4. **Một số lớp tiện ích trong Java**
5. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

44

Bài tập: Tính tổng số lượng các số nguyên bất kỳ

```
public class Test {  
    public static int plus(int... arr) {  
        int result = 0;  
        for (int i : arr) {  
            result += i;  
        }  
        return result;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(plus(1, 2, 3, 4, 5));  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

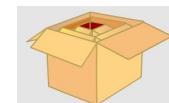
43

4.1. Package trong Java

❖ Package giống như thư mục giúp:

- Tổ chức và xác định vị trí lớp dễ dàng và sử dụng các lớp một cách phù hợp.
- Tránh cho việc đặt tên lớp bị xung đột (trùng tên)
 - Các package khác nhau có thể chứa các lớp có cùng tên
- Bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn so với mối quan hệ giữa các lớp.

❖ Một package cũng có thể chứa các package khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

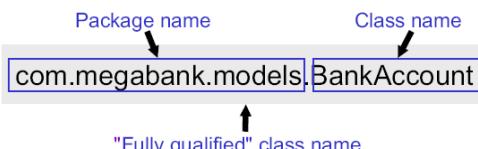
45

45

11

4.1. Package trong Java (2)

- Tên đầy đủ của lớp bao gồm tên gói và tên lớp:



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

46

a. Tham chiếu giữa các lớp

- Trong cùng 1 package: Sử dụng tên lớp
- Khác package: Phải cung cấp tên đầy đủ cho các lớp được định nghĩa trong package khác.
- Ví dụ:

```
public class HelloNameDialog {  
    public static void main(String[] args){  
        String result;  
        result = javax.swing.JOptionPane.  
            showInputDialog("Hay nhap ten ban:");  
        javax.swing.JOptionPane.showMessageDialog(null,  
            "Xin chao " + result + "!");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

47

a. Tham chiếu giữa các lớp (2)

❖ Lệnh import:

- Sử dụng lệnh **import** để khai báo các package hoặc các lớp để khi sử dụng không cần nêu tên đầy đủ.
- Ví dụ:

```
import javax.swing.JOptionPane;  
public class HelloNameDialog {  
    public static void main(String[] args){  
        String result;  
        result = JOptionPane.showInputDialog("Hay nhap ten ban:");  
        JOptionPane.showMessageDialog(null, "Xin chao " + result + "!");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

48

b. Các package trong Java

- java.applet
- java.awt
- java.beans
- java.io
- java.lang
- java.math
- java.net
- java.nio
- java.rmi
- java.security
- java.sql
- java.text
- java.util
- javax.accessibility
- javax.crypto
- javax.imageio
- javax.naming
- javax.net
- javax.print
- javax.rmi
- javax.sound
- javax.transaction
- javax.xml
- org.ietf.jgss
- org.omg.CORBA
- org.omg.CosNaming
- org.omg.Dynamic
- org.omg.IOP
- org.omg.Messaging
- org.omg.PortableInterceptor
- org.omg.PortableServer
- org.omg.SendingContext
- org.omg.stub.java.rmi
- org.w3c.dom
- org.xml



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

49

b. Các package trong Java (2)

❖ Các package cơ bản trong Java

▪ java.lang

- Cung cấp các lớp cơ bản cho thiết kế ngôn ngữ lập trình Java
- Bao gồm wrapper classes, String và StringBuffer, Object, ...
- Import ngầm định vào tất cả các lớp

▪ java.util

- Bao gồm tập hợp framework, mô hình sự kiện, date time, và nhiều tiện ích khác.

▪ java.io

- Cung cấp khả năng vào/ra hệ thống với các luồng dữ liệu và hệ thống file.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

50

4.2. Các lớp bao (Wrapper class)

- ❖ Các kiểu dữ liệu nguyên thủy không có các phương thức liên quan đến nó.
- ❖ Mỗi kiểu dữ liệu nguyên thủy có một lớp tương ứng gọi là lớp bao:
 - Các lớp bao sẽ “gói” dữ liệu nguyên thủy và cung cấp các phương thức thích hợp cho dữ liệu đó.
 - Mỗi đối tượng của lớp bao đơn giản là lưu trữ một biến đơn và đưa ra các phương thức để xử lý nó.
 - Các lớp bao là một phần của Java API



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

52

b. Các package trong Java (3)

❖ Các package cơ bản trong Java

▪ java.math

- Cung cấp các lớp thực thi các phép toán với số nguyên và các phép toán thập phân

▪ java.sql

- Cung cấp các API cho phép truy nhập và xử lý dữ liệu được lưu trữ trong một nguồn dữ liệu (thường sử dụng cơ sở dữ liệu quan hệ)

▪ javax.swing

- Cung cấp các lớp và giao diện cho phép tạo ra các ứng dụng đồ họa.

▪ ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

51

4.2. Các lớp bao (2)

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

13

a. Chuyển đổi kiểu dữ liệu

- ❖ Sử dụng `toString()` để chuyển các giá trị số thành xâu.
- ❖ Sử dụng `<type>Value()` để chuyển từ đối tượng của lớp bao thành giá trị nguyên thủy của đối tượng tương ứng

```
Float objF = new Float("4.67");
float f = objF.floatValue(); // f=4.67F
int i = objF.intValue(); //i=4

❖ Sử dụng parse<type>() và valueOf() để chuyển xâu thành các giá trị số.

int i = Integer.parseInt("123"); //i=123
double d = Double.parseDouble("1.5"); // d=1.5
Double objF2 = Double.valueOf("-36.12");
long l = objF2.longValue(); // l=-36L
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

b. Các hằng số

- ❖ Boolean
 - Boolean FALSE
 - Boolean TRUE
- ❖ Byte
 - byte MIN_VALUE
 - byte MAX_VALUE
- ❖ Character
 - int MAX_RADIX
 - char MAX_VALUE
 - int MIN_RADIX
 - char MIN_VALUE
 - Unicode classification constants
- ❖ Double
 - double MAX_VALUE
 - double MIN_VALUE
 - double NaN
 - double NEGATIVE_INFINITY
 - double POSITIVE_INFINITY

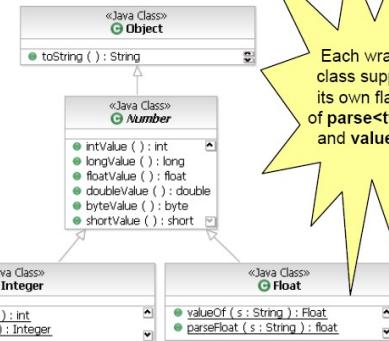
- ❖ Float
 - float MAX_VALUE
 - float MIN_VALUE
 - float NaN
 - float NEGATIVE_INFINITY
 - float POSITIVE_INFINITY
- ❖ Integer
 - int MIN_VALUE
 - int MAX_VALUE
- ❖ Long
 - long MIN_VALUE
 - long MAX_VALUE
- ❖ Short
 - short MIN_VALUE
 - short MAX_VALUE



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

a. Chuyển đổi kiểu dữ liệu (2)



Each wrapper class supports its own flavors of `parse<type>()` and `valueOf()`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

Ví dụ

```
double d = (new Integer(Integer.MAX_VALUE)).
            doubleValue();
System.out.println(d); // 2.147483647E9

String input = "test 1-2-3";
int output = 0;
for (int index=0;index<input.length();index++) {
    char c = input.charAt(index);
    if (Character.isDigit(c))
        output = output * 10 + Character.digit(c, 10);
}
System.out.println(output); // 123
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

4.3. Xâu (String)

- ❖ Kiểu String là một lớp và không phải là kiểu dữ liệu nguyên thủy
- ❖ Một String được tạo thành từ một dãy các ký tự nằm trong dấu nháy kép:

```
String a = "A String";  
String b = "";
```

- ❖ Đổi tượng String có thể khởi tạo theo nhiều cách:

```
String c = "A String";  
String d = new String("Another String");  
String e = null;
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

58

b. Các phương thức của xâu

```
String name = "Joe Smith";  
name.toLowerCase();           // "joe smith"  
name.toUpperCase();         // "JOE SMITH"  
"Joe Smith".trim();          // "Joe Smith"  
"Joe Smith".indexOf('e');    // 2  
"Joe Smith".length();        // 9  
"Joe Smith".charAt(5);       // 'm'  
"Joe Smith".substring(5);     // "mith"  
"Joe Smith".substring(2,5);    // "e S"
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

60

a. Ghép xâu

- ❖ Toán tử + có thể nối các String:

```
String a = "This" + " is a " + "String";  
//a = "This is a String"
```

- ❖ Các kiểu dữ liệu cơ bản sử dụng trong lời gọi println() được chuyển đổi tự động sang kiểu String

```
System.out.println("answer = " + 1 + 2 + 3);  
System.out.println("answer = " + (1+2+3));
```

→ Hai câu lệnh trên có in ra cùng một kết quả?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

59

c. So sánh hai xâu

- ❖ oneString.equals(anotherString)

- Kiểm tra tính tương đương
- Trả về true hoặc false

```
String name = "Joe";  
if ("Joe".equals(name))  
    name += " Smith";
```

- ❖ oneString.equalsIgnoreCase(anotherString)

- Kiểm tra KHÔNG xét đến ký tự hoa, thường

```
boolean same = "Joe".equalsIgnoreCase("joe");
```

- ❖ So sánh oneString == anotherString sẽ gây nhầm lẫn

- So sánh 2 đối tượng



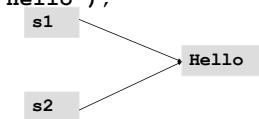
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

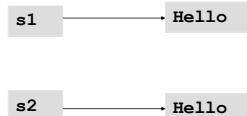
61

c. So sánh hai xâu (2)

```
String s1 = new String("Hello");
String s2 = s1;
(s1==s2) trả về true
```

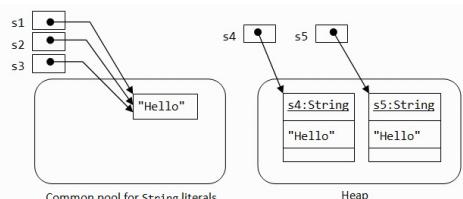


```
String s1 = new String("Hello");
String s2 = new String("Hello");
(s1==s2) trả về false
```



String Literal vs. String Object

- ❖ String s1 = "Hello"; // String literal
- ❖ String s2 = "Hello"; // String literal
- ❖ String s3 = s1; // same reference
- ❖ String s4 = new String("Hello"); // String object
- ❖ String s5 = new String("Hello"); // String object



d. Điểm đặc biệt của String

- ❖ Khởi tạo String theo 2 cách:

- Gán 1 giá trị literal
- Dùng toán tử new (Không khuyến khích dùng)

- ❖ Ví dụ:

- String str1 = "Java is Hot"; // **Implicit construction qua string literal**
 - str1 is được khai báo là 1 String reference và được khởi tạo 1 giá trị String literal "Java is Hot"
- String str2 = new String("I'm cool"); // **Explicit construction qua toán tử new**
 - str2 được khai báo là 1 String reference và được khởi tạo qua toán tử new.

- ❖ String literals được chứa trong **1 common pool**.

- → Cho phép chia sẻ lưu trữ các String với cùng nội dung để tiết kiệm bộ nhớ.

- ❖ String objects lưu trữ giá trị trong heap, không tiết kiệm được bộ nhớ



d. Điểm đặc biệt của String (2)

```
String s1 = new String("test");
String s2 = "test";
String s3 = String.valueOf("test");
System.out.println(s1==s2);
System.out.println(s1==s3);
System.out.println(s2==s3);
```

```
String s4 = new String("test");
String s5 = "test";
String s6 = String.valueOf("test");
System.out.println(s1==s4);
System.out.println(s2==s5);
System.out.println(s3==s6);
```



4.4. StringBuffer

❖ String là kiểu bất biến:

- Đối tượng không thay đổi giá trị sau khi được tạo ra → Các xâu của lớp String được thiết kế để không thay đổi giá trị.
- Khi các xâu được ghép nối với nhau một đối tượng mới được tạo ra để lưu trữ kết quả → Ghép nối xâu thông thường rất tốn kém về bộ nhớ.

❖ StringBuffer là kiểu biến đổi:

- Đối tượng có thể thay đổi giá trị sau khi được tạo ra

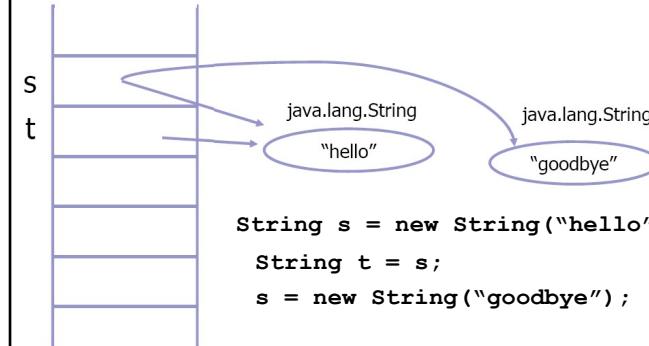


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

66

4.4. StringBuffer (2)



```
String s = new String("hello");
String t = s;
s = new String("goodbye");
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

67

67

4.4. StringBuffer (3)

❖ StringBuffer:

- Cung cấp các đối tượng xâu có thể thay đổi giá trị → Sử dụng **StringBuffer** khi:
 - Dự đoán các ký tự trong xâu có thể thay đổi.
 - Khi xử lý các xâu một cách linh động, ví dụ như đọc dữ liệu text từ một tệp tin.
- Cung cấp các cơ chế hiệu quả hơn cho việc xây dựng, ghép nối các xâu:
 - Việc ghép nối xâu thường được các trình biên dịch chuyển sang thực thi trong lớp StringBuffer



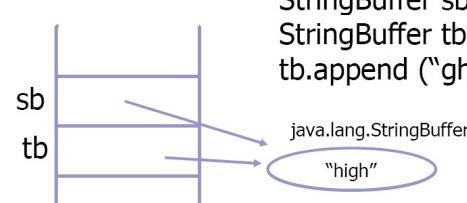
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

68

68

4.4. StringBuffer (4)

- ❖ Tính biến đổi: Nếu một đối tượng bị biến đổi, thì tất cả các quan hệ với đối tượng sẽ nhận giá trị mới.



```
StringBuffer sb = new ("hi");
StringBuffer tb = sb;
tb.append ("gh");
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

69

4.4. StringBuffer (5)

- ❖ Nếu tạo xâu thông qua vòng lặp thì sử dụng **StringBuffer**

```
StringBuffer buffer = new StringBuffer(15);
buffer.append("This is ");
buffer.append("String");
buffer.insert(7, " a");
buffer.append('.');
System.out.println(buffer.length()); // 17
System.out.println(buffer.capacity()); // 32
String output = buffer.toString();
System.out.println(output); // "This is a String."
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

4.5. Lớp Math (2)

- ❖ Hầu hết các hàm nhận tham số kiểu **double** và giá trị trả về cũng có kiểu **double**

- Ví dụ:

$$e^{\sqrt{2\pi}}$$

```
Math.pow(Math.E,
         Math.sqrt(2.0*Math.PI))
```

Hoặc:

```
Math.exp(Math.sqrt(2.0*Math.PI))
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

4.5. Lớp Math

- ❖ java.lang.Math cung cấp các thành phần static:

- Các hằng toán học:

- Math.E
- Math.PI

- Các hàm toán học:

- max, min...
- abs, floor, ceil...
- sqrt, pow, log, exp...
- cos, sin, tan, acos, asin, atan...
- random



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

1

71

4.6. Lớp System

- ❖ java.lang.System chứa nhiều hàm tiện ích hữu dụng

- Kiểm soát vào ra (I/O) chuẩn

- Các luồng InputStream in, PrintStreams out và err là các thuộc tính của lớp System.
- Có thể thiết lập lại nhờ các hàm setIn(), setOut() và setErr()

- **arraycopy ()**: Sao chép mảng hoặc tập con với hiệu năng cao.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

73

4.6. Lớp System (2)

- ❖ **currentTimeMillis ()** : Trả về thời gian hiện tại theo millisecond
- ❖ **exit ()** : Kết thúc hoạt động của Java Virtual Machine
- ❖ **gc ()** : Yêu cầu bộ thu gom rác hoạt động
- ❖ Các phương thức liên quan đến thuộc tính của hệ thống: Lấy các thông tin thuộc tính như phiên bản của Java Runtime Environment version, thư mục cài đặt Java,...

```
System.out.println(System.currentTimeMillis());
```



74

4.6. Lớp System (4)



```
C:\Windows\system32\cmd.exe
C:\Users\DatTT\Desktop\Java>java PropertiesTest
;
;
Windows 7
6.1
C:\Users\DatTT\Desktop\Java
C:\Users\DatTT
DatTT
C:\Users\DatTT\Desktop\Java>
```



76

4.6. Lớp System (3)

```
import java.util.Properties;

public class PropertiesTest {
    public static void main(String[] args) {
        System.out.println(System.getProperty("path.separator"));
        System.out.println(System.getProperty("file.separator"));
        System.out.println(System.getProperty("java.class.path"));
        System.out.println(System.getProperty("os.name"));
        System.out.println(System.getProperty("os.version"));
        System.out.println(System.getProperty("user.dir"));
        System.out.println(System.getProperty("user.home"));
        System.out.println(System.getProperty("user.name"));
    }
}
```



75

Nội dung

1. Chồng phương thức
2. Thành viên ĐT và thành viên lớp
3. Truyền tham số cho phương thức
4. Một số lớp tiện ích trong Java
5. Ví dụ và bài tập



77

19

Bài tập 1

- ❖ Tiếp bài tập 2 của bài học trước, sử dụng thành viên lớp để cài đặt đếm số đối tượng của lớp NhanVien được tạo ra tại bất kỳ thời điểm nào, ở bất kỳ đâu. Cài đặt minh họa cách thức sử dụng.

Bài tập 2

Bài tập 2

- ❖ Cài đặt phương thức, đầu vào là số lượng bất kỳ các đối tượng lớp NhanVien, đầu ra là tổng lương của các đối tượng này.

Bài tập 3

- ❖ Sử dụng lớp System đếm thời gian phép cộng xâu nhiều lần với String, và dùng phương thức append với StringBuffer để so sánh hiệu năng.



Bài 6: Kết tập và kế thừa

1

Mục tiêu bài học

- ❖ Giải thích về khái niệm tái sử dụng mã nguồn
- ❖ Chỉ ra được bản chất, mô tả các khái niệm liên quan đến kết tập và kế thừa
- ❖ So sánh kết tập và kế thừa
- ❖ Biểu diễn được kết tập và kế thừa trên UML
- ❖ Giải thích nguyên lý kế thừa và thứ tự khởi tạo, hủy bỏ đối tượng trong kế thừa
- ❖ Áp dụng các kỹ thuật, nguyên lý về kết tập và kế thừa trên ngôn ngữ lập trình Java



2

Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)
4. Ví dụ và bài tập



3

Nội dung

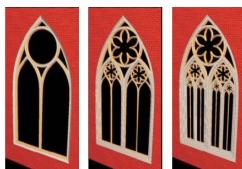
1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)
4. Ví dụ và bài tập



4

1. Tái sử dụng mã nguồn (Re-usability)

- ❖ Tái sử dụng mã nguồn: Sử dụng lại các mã nguồn đã viết
 - Lập trình cấu trúc: Tái sử dụng hàm/chương trình con
 - OOP: Khi mô hình thế giới thực, tồn tại nhiều loại đối tượng có các thuộc tính và hành vi tương tự hoặc liên quan đến nhau
 - → Làm thế nào để tái sử dụng lớp đã viết?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

Ưu điểm của tái sử dụng mã nguồn

- ❖ Giảm thiểu công sức, chi phí
- ❖ Nâng cao chất lượng phần mềm
- ❖ Nâng cao khả năng mô hình hóa thế giới thực
- ❖ Nâng cao khả năng bảo trì (maintainability)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

1. Tái sử dụng mã nguồn (2)

- ❖ Các cách sử dụng lại lớp đã có:
 - Sao chép lớp cũ thành 1 lớp khác → Dư thừa và khó quản lý khi có thay đổi
 - Tạo ra lớp mới là sự *tập hợp* hoặc *sử dụng các đối tượng* của lớp cũ đã có → Kết tập (Aggregation)
 - Tạo ra lớp mới trên cơ sở *phát triển* từ lớp cũ đã có → Kế thừa (Inheritance)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)
4. Ví dụ và bài tập



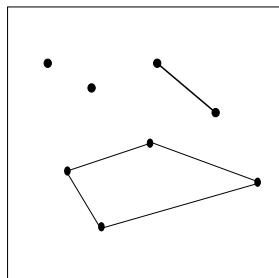
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

2. Kết tập

❖ Ví dụ:

- Điểm
 - Tứ giác gồm 4 điểm
→ Kết tập



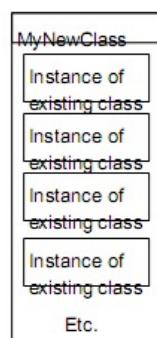
❖ Kết tập

- Quan hệ chứa/có ("has-a") hoặc là một phần (is-a-part-of)

2.1. Bản chất của kết tập (2)

❖ Lớp toàn thể chứa đối tượng của lớp thành phần

- Là một phần (is-a-part of) của lớp toàn thể
- Tái sử dụng các thành phần dữ liệu và các hành vi của lớp thành phần thông qua đối tượng thành phần



2.1. Bản chất của kết tập

❖ Kết tập (aggregation)

- Tao ra các đối tượng của các lớp có sẵn trong lớp mới → thành viên của lớp mới.
- Kết tập tái sử dụng thông qua *đối tượng*

❖ Lớp mới

- Lớp toàn thể (Aggregate/Whole),

❖ Lớp cũ

- Lớp thành phần (Part).

2.2. Biểu diễn kết tập bằng UML

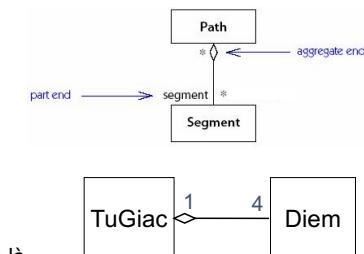
❖ Sử dụng "hình thoi" tại đầu của lớp toàn thể

❖ Sử dụng bộ số quan hệ (multiplicity) tại 2 đầu

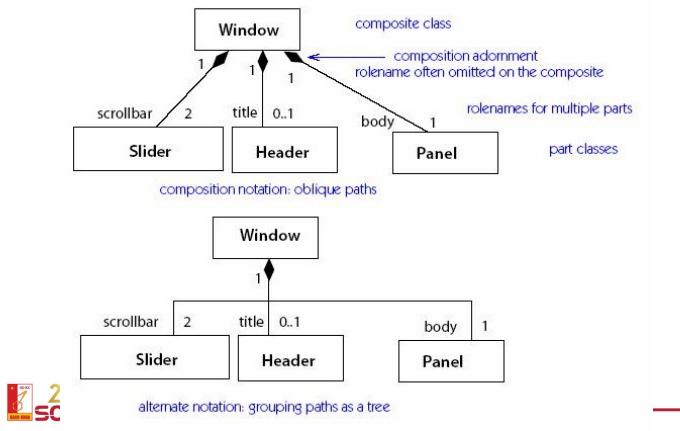
- 1 số nguyên dương: 1, 2, ...
- Dải số (0..1, 2..4)
- *: Bất kỳ số nào
- Không có: Mặc định là 1

❖ Tên vai trò (rolename)

- Nếu không có thì mặc định là tên của lớp (bỏ viết hoa chữ cái đầu)



Ví dụ



13

2.3. Minh họa trên Java

```
class Diem {  
    private int x, y;  
    public Diem(){ }  
    public Diem(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void setX(int x){ this.x = x; }  
    public int getX() { return x; }  
    public void printDiem(){  
        System.out.print("(" + x + ", " + y + ")");  
    }  
}
```

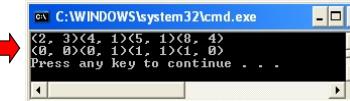


14

```
class TuGiac {  
    private Diem d1, d2;  
    private Diem d3, d4;  
    public TuGiac(Diem p1, Diem p2,  
                  Diem p3, Diem p4){  
        d1 = p1; d2 = p2; d3 = p3; d4 = p4;  
    }  
    public TuGiac(){  
        d1 = new Diem(); d2 = new Diem(0,1);  
        d3 = new Diem(1,1); d4 = new Diem(1,0);  
    }  
    public void printTuGiac(){  
        d1.printDiem(); d2.printDiem();  
        d3.printDiem(); d4.printDiem();  
        System.out.println();  
    }  
}
```

15

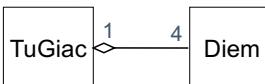
```
public class Test {  
    public static void main(String arg[]){  
        Diem d1 = new Diem(2,3);  
        Diem d2 = new Diem(4,1);  
        Diem d3 = new Diem(5,1);  
        Diem d4 = new Diem(8,4);  
  
        TuGiac tg1 = new TuGiac(d1, d2, d3, d4);  
        TuGiac tg2 = new TuGiac();  
        tg1.printTuGiac();  
        tg2.printTuGiac();  
    }  
}
```



16

Cách cài đặt khác

```
class TuGiac {  
    private Diem[] diem = new Diem[4];  
    public TuGiac(Diem p1, Diem p2,  
                  Diem p3, Diem p4){  
        diem[0] = p1; diem[1] = p2;  
        diem[2] = p3; diem[3] = p4;  
    }  
    public void printTuGiac(){  
        diem[0].printDiem(); diem[1].printDiem();  
        diem[2].printDiem(); diem[3].printDiem();  
        System.out.println();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

2.4. Thứ tự khởi tạo trong kết tập

- ❖ Khi một đối tượng được tạo mới, các thuộc tính của đối tượng đó đều phải được khởi tạo và gán những giá trị tương ứng.
- ❖ Các đối tượng thành phần được khởi tạo trước
→ Các phương thức khởi tạo của các lớp của các đối tượng thành phần được thực hiện trước



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. **Kế thừa (Inheritance)**
4. Ví dụ và bài tập

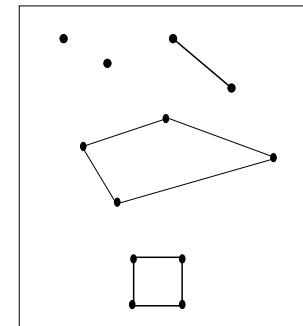


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

3.1. Tổng quan về kế thừa

- ❖ Ví dụ:
 - Điểm
 - Tứ giác gồm 4 điểm
 - Kết tập
 - Tứ giác
 - Hình vuông
 - Kế thừa



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

3.1.1. Bản chất kế thừa

❖ Kế thừa (Inherit, Derive)

- Tạo lớp mới bằng cách phát triển lớp đã có.
- Lớp mới kế thừa những gì đã có trong lớp cũ và phát triển những tính năng mới.

❖ Lớp cũ:

- Lớp cha (parent, superclass), lớp cơ sở (base class)

❖ Lớp mới:

- Lớp con (child, subclass), lớp dẫn xuất (derived class)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

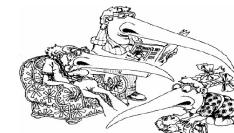
21

21

3.1.1. Bản chất kế thừa (2)

❖ Lớp con

- Là một loại (is-a-kind-of) của lớp cha
- Tái sử dụng bằng cách kế thừa các thành phần dữ liệu và các hành vi của lớp cha
- Chi tiết hóa cho phù hợp với mục đích sử dụng mới
 - Extension: Thêm các thuộc tính/hành vi mới
 - Redefinition (Method Overriding): Chính sửa lại các hành vi kế thừa từ lớp cha



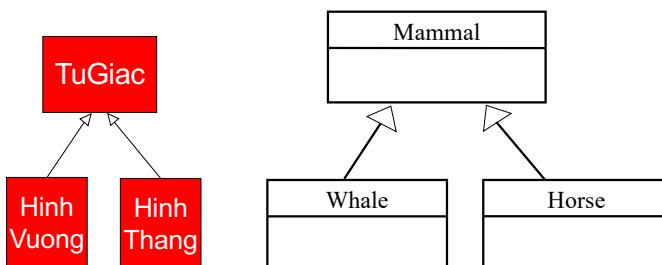
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

3.1.2. Biểu diễn kế thừa trong UML

❖ Sử dụng "tam giác rỗng" tại đầu Lớp cha



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

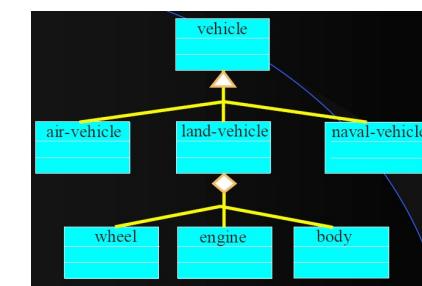
23

23

3.1.3. Kết tập và kế thừa

❖ So sánh kết tập và kế thừa?

- Giống nhau
 - Đều là kỹ thuật trong OOP để tái sử dụng mã nguồn
- Khác nhau?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

24

Phân biệt kế thừa và kết tập

Kế thừa

- ❖ Kế thừa **tái sử dụng thông qua lớp**
 - Tạo lớp mới bằng cách phát triển lớp đã có
 - Lớp con kế thừa dữ liệu và hành vi của lớp cha
- ❖ Quan hệ "là một loại" ("is a kind of")
- ❖ Ví dụ: Ô tô là một loại phương tiện vận tải

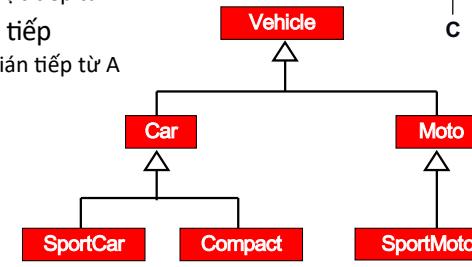
Kết tập

- ❖ Kết tập **tái sử dụng thông qua đối tượng**.
 - Tạo ra lớp mới là tập hợp các đối tượng của các lớp đã có
 - Lớp toàn thể có thể sử dụng dữ liệu và hành vi thông qua các đối tượng thành phần
- ❖ Quan hệ "là một phần" ("is a part of")
- ❖ Ví dụ: Bánh xe là một phần của Ô tô

25

3.1.4. Cây phân cấp kế thừa (Inheritance hierarchy)

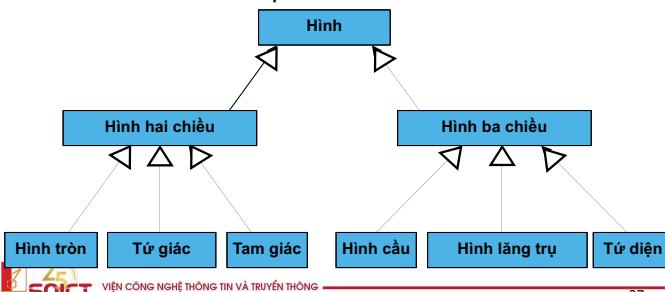
- ❖ Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
- ❖ Dẫn xuất trực tiếp
 - B dẫn xuất trực tiếp từ A
- ❖ Dẫn xuất gián tiếp
 - C dẫn xuất gián tiếp từ A



26

3.1.4. Cây phân cấp kế thừa (2)

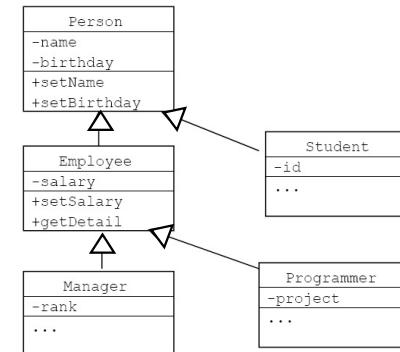
- ❖ Các lớp con có cùng lớp cha gọi là anh chị em (siblings)
- ❖ Thành viên được kế thừa sẽ được kế thừa xuống dưới trong cây phân cấp → Lớp con kế thừa tất cả các lớp tổ tiên của nó



27

3.1.4. Cây phân cấp kế thừa (2)

Mọi lớp
đều kế thừa từ
lớp gốc Object



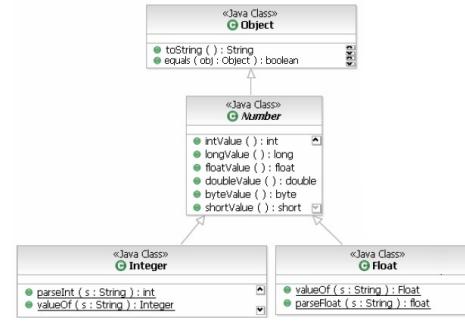
28

Lớp Object

- ❖ Trong gói `java.lang`
- ❖ Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp `Object`.
→ Lớp `Object` là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa

Lớp Object (2)

- ❖ Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp, ví dụ: `toString()`, `equals()`...



3.2. Nguyên lý kế thừa

- ❖ Chỉ định truy cập `protected`
- ❖ Thành viên `protected` trong lớp cha được truy cập trong:
 - Các thành viên lớp cha
 - **Các thành viên lớp con**
 - Các thành viên các lớp cùng thuộc 1 package với lớp cha
- ❖ Lớp con có thể kế thừa được gì?
 - Kế thừa được các thành viên được khai báo là `public` và `protected` của lớp cha.
 - Không kế thừa được các thành viên `private`.
 - Các thành viên có chỉ định truy cập mặc định nếu lớp cha cùng gói với lớp con

3.2. Nguyên lý kế thừa (2)

	<code>public</code>	Không có	<code>protected</code>	<code>private</code>
Cùng lớp cha				
Lớp con cùng gói				
Lớp con khác gói				
Khác gói, non-inher				

3.2. Nguyên lý kế thừa (2)

	public	Không có	protected	private
Cùng lớp cha	Yes	Yes	Yes	Yes
Lớp con cùng gói	Yes	Yes	Yes	No
Lớp con khác gói	Yes	No	Yes	No
Khác gói, non-inher	Yes	No	No	No



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

3.2. Nguyên lý kế thừa (3)

❖ Các trường hợp không được phép kế thừa:

- Các phương thức khởi tạo và hủy
 - Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
 - Chúng chỉ biết cách làm việc với từng lớp cụ thể
- Toán tử gán =
 - Làm nhiệm vụ giống như phương thức khởi tạo



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

3.3. Cú pháp kế thừa trên Java

- ❖ Cú pháp kế thừa trên Java:
 - <Lớp con> extends <Lớp cha>
- ❖ Lớp cha nếu được định nghĩa là **final** thì không thể có lớp dẫn xuất từ nó.
- ❖ Ví dụ:

```
class HinhVuong extends TuGiac {  
    ...  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

```
public class TuGiac {  
    protected Diem d1, d2, d3, d4;  
    public void setD1(Diem _d1) {_d1=_d1;}  
    public Diem getD1(){return d1;}  
    public void printTuGiac(){...}  
    ...  
}
```

Ví dụ 1

Sử dụng các thuộc tính
protected của lớp cha
trong lớp con

```
public class HinhVuong extends TuGiac {  
    public HinhVuong(){  
        d1 = new Diem(0,0); d2 = new Diem(0,1);  
        d3 = new Diem(1,0); d4 = new Diem(1,1);  
    }  
}  
public class Test{  
    public static void main(String args[]){  
        HinhVuong hv = new HinhVuong();  
        hv.printTuGiac();  
    }  
}
```

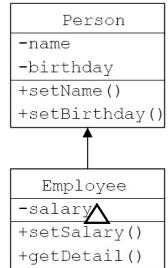
Gọi phương thức public
lớp cha của đối tượng lớp con

36

35

Ví dụ 2

```
protected  
class Person {  
    private String name;  
    private Date birthday;  
    public String getName() {return name;}  
    ...  
}  
class Employee extends Person {  
    private double salary;  
    public boolean setSalary(double sal){  
        salary = sal;  
        return true;  
    }  
    public String getDetail(){  
        String s = name+", "+birthday+", "+salary; //Loi  
    }  
}
```

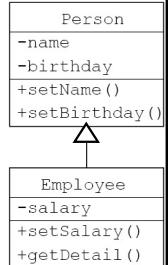


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

Ví dụ 2

```
protected  
class Person {  
    protected String name;  
    protected Date birthday;  
    public String getName() {return name;}  
    ...  
}  
class Employee extends Person {  
    private double salary;  
    public boolean setSalary(double sal){  
        salary = sal;  
        return true;  
    }  
    public String getDetail(){  
        String s = name+", "+birthday+", "+salary;  
    }  
}
```

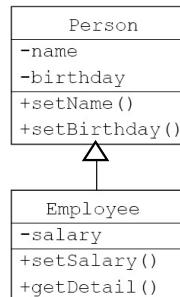


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

Ví dụ 2 (tiếp)

```
public class Test {  
    public static void main(String args[]){  
        Employee e = new Employee();  
        e.setName("John");  
        e.setSalary(3.0);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

Ví dụ 3 – Cùng gói

```
public class Person {  
    Date birthday;  
    String name;  
    ...  
}  
public class Employee extends Person {  
    ...  
    public String getDetail() {  
        String s;  
        String s = name + "," + birthday;  
        s += ", " + salary;  
        return s;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

Ví dụ 3 – Khác gói

```
package abc;
public class Person {
    protected Date birthday;
    protected String name;
    ...
}

import abc.Person;
public class Employee extends Person {
    ...
    public String getDetail() {
        String s;
        s = name + "," + birthday + "," + salary;
        return s;
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

3.4. Khởi tạo và huỷ bỏ đối tượng

❖ Khởi tạo đối tượng:

- Lớp cha được khởi tạo trước lớp con.
- Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên
 - Tự động gọi (không tường minh - implicit): Khi lớp cha CÓ phương thức khởi tạo mặc định
 - Gọi trực tiếp (tường minh - explicit)

❖ Hủy bỏ đối tượng:

- Ngược lại so với khởi tạo đối tượng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

3.4.1. Tự động gọi constructor của lớp cha

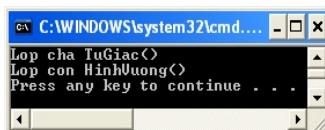
```
public class TuGiac {
    protected Diem d1, d2;
    protected Diem d3, d4;
    public TuGiac() {
        System.out.println
            ("Lop cha TuGiac()");
    }
    //...
}

public class HinhVuong
    extends TuGiac {
    public HinhVuong() {
        //Tu dong goi TuGiac()
        System.out.println
            ("Lop con HinhVuong()");
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

44

3.4.2. Gọi trực tiếp constructor của lớp cha

❖ Câu lệnh đầu tiên trong phương thức khởi tạo của lớp con gọi phương thức khởi tạo của lớp cha

- super (Danh_sach_tham_so);
- Điều này là bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
 - Đã viết phương thức khởi tạo của lớp cha với một số tham số
 - Phương thức khởi tạo của lớp con không bắt buộc phải có tham số.

Ví dụ

```
public class TuGiac {  
    protected Diem d1, d2;  
    protected Diem d3, d4;  
    public TuGiac(Diem d1,  
                 Diem d2, Diem d3, Diem d4){  
        System.out.println("Lop cha  
        TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong(){  
        System.out.println  
        ("Lop con HinhVuong()");  
    }  
}
```

```
public class Test {  
    public static void  
    main(String arg[]){  
        HinhVuong hv =  
        new  
        HinhVuong();  
    }  
}
```

Lỗi



```
D:\FIT-HUT\Lessons\OOP\Lesson 1>javac -c  
HinhVuong.java  
HinhVuong.java:8: cannot find symbol  
symbol : constructor TuGiac()  
location: class TuGiac  
    public HinhVuong(){  
           ^  
1 error
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

Gọi trực tiếp constructor của lớp cha

Phương thức khởi tạo lớp con KHÔNG tham số

```
public class TuGiac {  
    protected Diem d1,d2,d3,d4;  
    public TuGiac(Diem d1, Diem d2,  
                 Diem d3, Diem d4){  
        System.out.println("Lop cha  
        TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong(){  
        super(new Diem(0,0), new Diem(0,1),  
              new Diem(1,1),new Diem(1,0));  
        System.out.println("Lop con HinhVuong()");  
    }  
}
```



```
C:\WINDOWS\system32\cmd.exe  
Lop cha TuGiac(d1, d2, d3, d4)  
Lop con HinhVuong()  
Press any key to continue . . .
```



46

Gọi trực tiếp constructor của lớp cha Phương thức khởi tạo lớp con CÓ tham số

```
public class TuGiac {  
    protected Diem d1,d2,d3,d4;  
    public TuGiac(Diem d1,  
                 Diem d2, Diem d3, Diem d4){  
        System.out.println  
        ("Lop cha TuGiac(d1,d2,d3,d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong(Diem d1, Diem d2,  
                     Diem d3, Diem d4){  
        super(d1, d2, d3, d4);  
        System.out.println("Lop con HinhVuong(d1,d2,d3,d4)");  
    }  
}
```

Lỗi

```
Lop cha TuGiac(d1, d2, d3, d4)  
Lop con HinhVuong(d1, d2, d3, d4)
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

Ví dụ

```
public class TG1 {  
    private String name;  
    public TG1(String name){  
    }  
}  
  
public class TG2 {  
    private String name;  
    public TG2(String name){  
    }  
}  
  
public class HV1 extends TG1 {  
    public HV1(){  
    }  
    public void test(){  
    }  
}  
  
public class HV2 extends TG2 {  
    public void test(){  
    }  
}
```

Lớp HV1, HV2 bị lỗi biên dịch?



```
D:\FIT-HUT\Lessons\OOP\Lesson 1>javac -c  
HV1.java  
HV1.java:2: error: cannot find symbol  
symbol : class TG1  
location: class HV1  
    public class HV1 extends TG1 {  
                           ^  
1 error
```

48

12

Nội dung

1. Tái sử dụng mã nguồn
2. Kết tập (Aggregation)
3. Kế thừa (Inheritance)
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

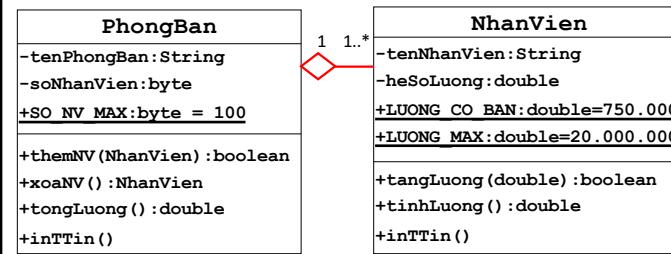
50

```
public class PhongBan {  
    private String tenPhongBan; private byte soNhanVien;  
    public static final SO_NV_MAX = 100;  
    private NhanVien[] dsnv;  
    public boolean themNhanVien(NhanVien nv){  
        if (soNhanVien < SO_NV_MAX) {  
            dsnv[soNhanVien] = nv; soNhanVien++;  
            return true;  
        } else return false;  
    }  
    public NhanVien xoaNhanVien(){  
        if (soNhanVien > 0) {  
            NhanVien tmp = dsnv[soNhanVien-1];  
            dsnv[soNhanVien-1] = null; soNhanVien--;  
            return tmp;  
        } else return null;  
    }  
    // (cont)...
```

52

Bài tập:

- ❖ Viết mã nguồn cho lớp **PhongBan** với các thuộc tính và phương thức như biểu đồ trên cùng phương thức khởi tạo với số lượng tham số cần thiết, biết rằng:
- Việc thêm/xóa nhân viên được thực hiện theo cơ chế của stack
 - **tongLuong()** trả về tổng lương của các nhân viên trong phòng.
 - **inTTin()** hiển thị thông tin của phòng và thông tin của các nhân viên trong phòng.



51

```
// (cont.)  
public PhongBan(String tenPB){  
    dsnv = new NhanVien[SO_NV_MAX];  
    tenPhongBan = tenPB; soNhanVien = 0;  
}  
public double tongLuong(){  
    double tong = 0.0;  
    for (int i=0;i<soNhanVien;i++)  
        tong += dsnv[i].tinhLuong();  
    return tong;  
}  
public void inTTin(){  
    System.out.println("Ten phong: "+tenPhong);  
    System.out.println("So NV: "+soNhanVien);  
    System.out.println("Thong tin cac NV");  
    for (int i=0;i<soNhanVien;i++)  
        dsnv[i].inTTin();  
}
```

53

Thảo luận

Trong ví dụ trên

❖ Lớp cũ? Lớp mới?

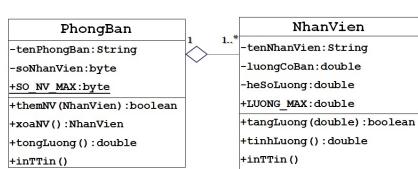
- Lớp cũ: NhanVien
- Lớp mới: PhongBan

❖ Lớp mới tái sử dụng lớp cũ thông qua?

- Mảng đối tượng của lớp NhanVien: dsnv

❖ Lớp mới tái sử dụng được những gì của lớp cũ?

- tinhLuong() trong phương thức tongLuong()
- inTTin() trong phương thức inTTin()





Bài 7: Một số kỹ thuật trong kế thừa

1

Mục tiêu

- ❖ Trình bày nguyên lý định nghĩa lại trong kế thừa
- ❖ Phân biệt khái niệm đơn kế thừa và đa kế thừa
- ❖ Giới thiệu về giao diện, lớp trừu tượng và vai trò của chúng
- ❖ Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



2

Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



3

Nội dung

1. **Định nghĩa lại/ghi đè (Overriding)**
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



4

1. Định nghĩa lại/ghi đè (Overriding)

❖ Quan hệ kế thừa (inheritance)

- Lớp con là một loại (is-a-kind-of) của lớp cha
- Kế thừa các thành phần dữ liệu và các hành vi của lớp cha
- Chi tiết hóa cho phù hợp với mục đích sử dụng mới:
 - Mở rộng lớp cha (Extension): Thêm các thuộc tính/hành vi mới
 - Định nghĩa lại (Redefinition): Chính sửa lại các hành vi kế thừa từ lớp cha → Ghi đè (Method Overriding)

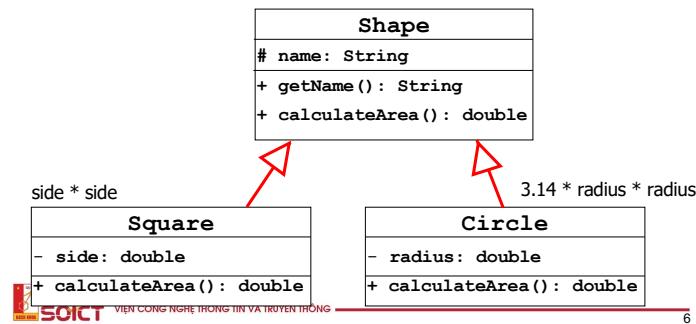


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1. Định nghĩa lại/ghi đè (Overriding)

- ❖ Phương thức ghi đè sẽ thay thế hoặc làm rõ hơn cho phương thức cùng tên trong lớp cha
- ❖ Đối tượng của lớp con sẽ hoạt động với phương thức mới phù hợp với nó



6

1. Định nghĩa lại/ghi đè (Overriding)

- ❖ Cú pháp: Phương thức ở lớp con hoàn toàn giống về chữ ký với phương thức kế thừa ở lớp cha
 - Trùng tên & danh sách tham số
 - Mục đích: Để thể hiện cùng bản chất công việc
- ❖ Lớp con có thể định nghĩa phương thức trùng tên với phương thức trong lớp cha:

Nếu phương thức mới chỉ trùng tên và khác chữ ký (số lượng hay kiểu dữ liệu của đối số)
→ Chồng phương thức (**Method Overloading**)

Nếu phương thức mới hoàn toàn giống về giao diện (chữ ký)
→ Định nghĩa lại hoặc ghi đè phương thức (**Method Override**)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

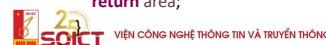
7

Ví dụ (1)

```
class Shape {
    protected String name;
    Shape(String n) { name = n; }
    public String getName() { return name; }
    public double calculateArea() { return 0.0; }
}

class Circle extends Shape {
    private double radius;
    Circle(String n, double r){
        super(n);
        radius = r;
    }

    public double calculateArea() {
        double area = (double) (3.14 * radius * radius);
        return area;
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

Ví dụ (2)

```
class Square extends Shape {  
    private double side;  
  
    Square(String n, double s) {  
        super(n);  
        side = s;  
    }  
    public double calculateArea() {  
        double area = (double) side * side;  
        return area;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

Thêm lớp Triangle

```
class Triangle extends Shape {  
    private double base, height;  
  
    Triangle(String n, double b, double h) {  
        super(n);  
        base = b;  
        height = h;  
    }  
    public double calculateArea() {  
        double area = 0.5f * base * height;  
        return area;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Muốn gọi lại các phương thức
của lớp cha đã bị ghi đè ?

10

10

Ví dụ sử dụng từ khóa super

```
public class Person {  
    protected String name;  
    protected int age;  
  
    public String getDetail() {  
        String s = this.name + "," + this.age;  
        return s;  
    }  
}  
  
public class Employee extends Person {  
    double salary;  
  
    public String getDetail() {  
        String s = super.getDetail() + "," + this.salary;  
        return s;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

Sử dụng từ khóa super

- ❖ Từ khóa super: tái sử dụng các đoạn mã của lớp cha trong lớp con
- ❖ Gọi phương thức khởi tạo
 - super(danh sách tham số);
- ❖ Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
- ❖ Gọi các phương thức của lớp cha
 - super.tên_Phương_thức(danh sách tham số);



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

Quy định trong ghi đè

- ❖ Phương thức ghi đè trong lớp con phải
 - Có danh sách tham số giống hệt phương thức kế thừa trong lớp cha.
 - Có cùng kiểu trả về với phương thức kế thừa trong lớp cha
 - Có chỉ định truy cập không giới hạn chặt hơn phương thức trong lớp cha. Ví dụ, nếu ghi đè một phương thức `protected`, thì phương thức mới có thể là `protected` hoặc `public`, mà không được là `private`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Quy định trong ghi đè

- ❖ Không được phép ghi đè:
 - Các phương thức `static` trong lớp cha
 - Các phương thức `private` trong lớp cha
 - Các phương thức hằng (`final`) trong lớp cha



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Ví dụ

```
class Parent {  
    public void doSomething() {}  
    protected int doSomething2() {  
        return 0;  
    }  
}  
  
class Child extends Parent {  
    protected void doSomething() {}  
    protected void doSomething2() {}  
}
```

cannot override: attempting to use incompatible return type

cannot override: attempting to assign weaker access privileges; was public



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Hạn chế ghi đè – Từ khóa `final`

- ❖ Đôi lúc ta muốn hạn chế việc định nghĩa lại vì các lý do sau:
 - Tính đúng đắn: Định nghĩa lại một phương thức trong lớp dẫn xuất có thể làm sai lạc ý nghĩa của nó
 - Tính hiệu quả: Cơ chế kết nối động không hiệu quả về mặt thời gian bằng kết nối tĩnh
- ❖ Nếu biết trước sẽ không định nghĩa lại phương thức của lớp cơ sở thì nên dùng từ khóa `final` đi với phương thức. Ví dụ:

```
public final String baseName () {  
    return "Person";  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

Hạn chế ghi đè – Từ khóa final

- ❖ Các phương thức được khai báo là final không thể ghi đè

```
class A {  
    final void method() {}  
}  
  
class B extends A{  
    void method(){ // Báo lỗi!!!  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

Hạn chế ghi đè – Từ khóa final

- ❖ Từ khóa **final** được dùng khi khai báo lớp:

- Lớp được khai báo là lớp hằng (không thay đổi), lớp này không có lớp con thừa kế
- Được sử dụng để hạn chế việc thừa kế và ngăn chặn việc sửa đổi một lớp

```
public final class A {  
    //...  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

this và super

- ❖ this và super có thể sử dụng cho các phương thức/thuộc tính non-static và phương thức khởi tạo
 - **this**: tìm kiếm phương thức/thuộc tính trong lớp hiện tại
 - **super**: tìm kiếm phương thức/thuộc tính trong lớp cha trực tiếp

- ❖ Từ khóa **super** cho phép tái sử dụng các đoạn mã của lớp cha trong lớp con



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

Bài tập 1

- ❖ Cho đoạn mã dưới đây:

```
1. class BaseClass {  
2.     private float x = 1.0f;  
3.     float getVar() { return x; }  
4. }  
5. class SubClass extends BaseClass {  
6.     private float x = 2.0f;  
7.     // insert code here  
8. }
```

- ❖ Lựa chọn nào có thể chèn tại dòng 7?

```
1. public double getVar() { return x; }  
2. public float getVar(float f){ return f; }  
3. float getVar() { return x; }  
4. public float getVar() { return x; }  
5. private float getVar() { return x; }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

19

Bài tập 2



❖ Cho đoạn mã dưới đây:

```
1. class Super {  
2.     public String getName() { return "Super"; }  
3. }  
4. class Sub extends Super {  
5.  
6. }
```

❖ Lựa chọn nào khi đặt vào dòng 5 trong đoạn mã trên **gây ra lỗi biên dịch?**

```
1. public String getTen () { }  
2. public void getName(String str) { }  
3. public String getName() {return "Sub"; }  
4. public void getName() { }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

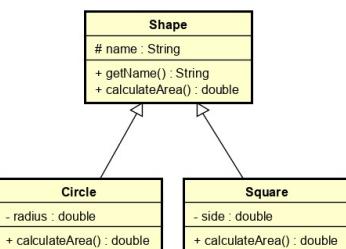
2. Lớp trừu tượng

❖ Các ngôn ngữ lập trình hướng đối tượng cung cấp các cơ chế kiểu trừu tượng (abstract type)

- Các kiểu trừu tượng có cài đặt không đầy đủ hoặc không có cài đặt
- Nhiệm vụ chính của chúng là giữ vai trò kiểu tổng quát hơn của một số các kiểu khác

❖ Xét ví dụ: lớp Shape

- Là một lớp "không rõ ràng", khó hình dung ra các đối tượng cụ thể
 - Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. **Lớp trừu tượng**
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

2. Lớp trừu tượng

❖ **Đặc điểm của lớp trừu tượng**

- Không thể tạo đối tượng trực tiếp từ các lớp trừu tượng
- Thường lớp trừu tượng được dùng để định nghĩa các "khái niệm chung", đóng vai trò làm lớp cơ sở (base class) cho các lớp "cụ thể" khác (concrete class)
- Chưa đầy đủ, thường được sử dụng làm lớp cha. Lớp con kế thừa nó sẽ hoàn thiện nó.
 - Lớp trừu tượng thường chứa các **phương thức trừu tượng** (phương thức không được cài đặt)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

24

2. Lớp trừu tượng

❖ Phương thức trừu tượng

- Là các phương thức “không rõ ràng” / chưa hoàn thiện, khó đưa ra cách cài đặt cụ thể
- Chỉ có chữ ký mà không có cài đặt cụ thể
- Các lớp dẫn xuất có thể làm rõ - định nghĩa lại (overriding) các phương thức trừu tượng này

Từ khóa abstract

❖ Lớp trừu tượng

- Khai báo với từ khóa **abstract**
`public abstract class Shape {
 // Nội dung lớp
}`

❖ Phương thức trừu tượng

- Khai báo với từ khóa **abstract**
`public abstract float calculateArea();`

`Shape = new Shape(); //Compile error`

Ví dụ Lớp trừu tượng

```
abstract class Shape {  
    protected String name;  
    Shape(String n) { name = n; }  
    public String getName() { return name; }  
    public abstract double calculateArea();  
}  
  
class Circle extends Shape {  
    private double radius;  
    Circle(String n, double r){  
        super(n);  
        radius = r;  
    }  
  
    public double calculateArea() {  
        double area = (double) (3.14 * radius * radius);  
        return area;  
    }  
}
```

Lớp con bắt buộc phải override tất cả các phương thức abstract của lớp cha

2. Lớp trừu tượng

❖ Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng

❖ Lớp con khi kế thừa phải cài đặt cụ thể cho các phương thức trừu tượng của lớp cha

- Nếu không ghi đè các phương thức này thì lớp con cũng trở thành một lớp trừu tượng → Phương thức trừu tượng không thể khai báo là **final** hoặc **static**

Kết hợp cho phép
abstract **public**
abstract **protected**

Kết hợp KHÔNG cho phép
abstract **private**
abstract **static**
abstract **final**



Ví dụ lớp trừu tượng

```
abstract class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void move(int dx, int dy) {  
        x += dx; y += dy;  
        plot();  
    }  
    public abstract void plot();  
    // phương thức trừu tượng không có  
    // phần code thực hiện  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

Ví dụ Lớp trừu tượng

```
abstract class ColoredPoint extends Point {  
    int color;  
    public ColoredPoint(int x, int y, int color) {  
        super(x, y); this.color = color;  
    }  
}  
  
class SimpleColoredPoint extends ColoredPoint {  
    public SimpleColoredPoint(int x, int y, int color) {  
        super(x, y, color);  
    }  
    @Override  
    public void plot() { ... }  
    // code to plot a SimplePoint  
}
```



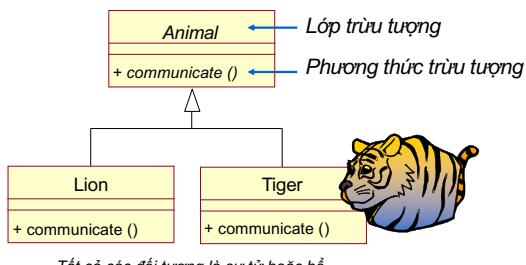
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

2. Lớp trừu tượng

❖ Biểu diễn trong UML

- Lớp trừu tượng (không thể tạo đối tượng cụ thể)
 - Chứa phương thức trừu tượng
 - Tên lớp / tên phương thức: Chữ nghiêng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

Bài tập 3

❖ 1. Đoạn mã dưới đây có lỗi gì không?

```
abstract class ABC {  
    void firstMethod() {  
        System.out.println("First Method");  
    }  
    void secondMethod() {  
        System.out.println("Second Method");  
    }  
}
```

❖ 2. Lớp nào là lớp trừu tượng, lớp nào có thể tạo đối tượng?

```
abstract class A { }  
  
class B extends A { }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

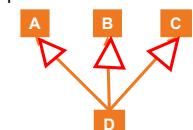
Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. **Đơn kế thừa & Đa kế thừa**
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập

3. Đơn kế thừa & Đa kế thừa

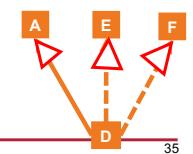
Đa kế thừa (Multiple Inheritance)

- Một lớp có thể kế thừa nhiều lớp cha trực tiếp
- C++ hỗ trợ đa kế thừa



Đơn kế thừa (Single Inheritance)

- Một lớp chỉ được kế thừa từ một lớp cha trực tiếp
- Java chỉ hỗ trợ đơn kế thừa
- → Đưa thêm khái niệm Giao diện (Interface)

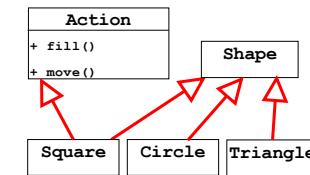


3. Đơn kế thừa & Đa kế thừa

❖ Giả sử trong bài toán các lớp đối tượng Hình học, lớp Square cần thiết kế bổ sung thêm những hành vi mới Fill (tô màu), Move (di chuyển) mà chỉ có các đối tượng của nó sử dụng

- Giải pháp 1: thêm các hành vi này vào lớp cha Shape → ảnh hưởng đến các đối tượng của lớp con Circle và Triangle (các đối tượng này không sử dụng đến các hành vi trên)
- Giải pháp 2: đặt các hành vi này trực tiếp tại lớp Square → tương lai có thể có thêm lớp mới Hình thang cũng sử dụng các hành vi trên → cần phải cài đặt lại, không tái sử dụng

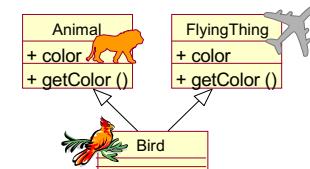
→ Tách các hành vi trên thành một lớp cha riêng, rồi cho lớp Square kế thừa cả HAI lớp cha trong cây thừa kế?



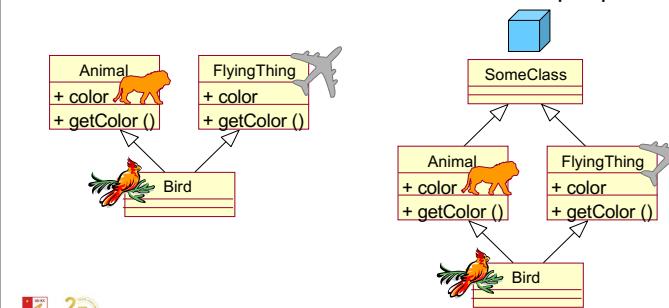
3. Đơn kế thừa & Đa kế thừa

Vấn đề gặp phải trong đa kế thừa

Name collision



"Diamond shape" problem



Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. **Giao diện (Interface)**
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

4. Giao diện

- ❖ Sử dụng từ khóa **interface** để định nghĩa
 - Một giao diện chỉ được bao gồm:
 - Chữ ký các phương thức (method signature)
 - Các thuộc tính khai báo hằng (static & final)
 - Không có thể hiện
 - Chỉ được thực thi và mở rộng
- ❖ Cú pháp khai báo giao diện trên Java

```
interface <Tên giao diện> { }
<Giao diện con> extends <Giao diện cha> { }
```
- ❖ Ví dụ

```
public interface DoiXung {...}
public interface Can extends DoiXung {...}
public interface DiChuyen {...}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

4. Giao diện

- ❖ Giao diện (interface)
 - Là kiểu dữ liệu trừu tượng, được dùng để đặc tả các hành vi mà các lớp phải thực thi
 - Tương tự như giao thức (protocols)
 - Chứa các chữ ký phương thức (Mỗi phương thức đều là phương thức trừu tượng) và các hằng
 - Giải quyết bài toán đa thừa kế, tránh các rắc rối nhập nhằng ngữ nghĩa
- ❖ Giao diện trong Java
 - Một cấu trúc lập trình của Java được định nghĩa với từ khóa **interface**
 - Từ Java 8: có thêm phương thức default, static. Từ Java 9, có thêm phương thức private và private static



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

4. Giao diện

- ❖ Lớp thực thi giao diện
 - Hoặc là lớp trừu tượng (abstract class)
 - Hoặc là bắt buộc phải cài đặt chi tiết toàn bộ các phương thức trong giao diện nếu là lớp cụ thể
- ❖ Một lớp có thể thực thi nhiều giao diện
`<Lớp con> [extends <Lớp cha>] implements <Đanh sách giao diện>`
- ❖ Ví dụ:

```
public class HinhVuong extends TuGiac
    implements DoiXung, DiChuyen {
}
```

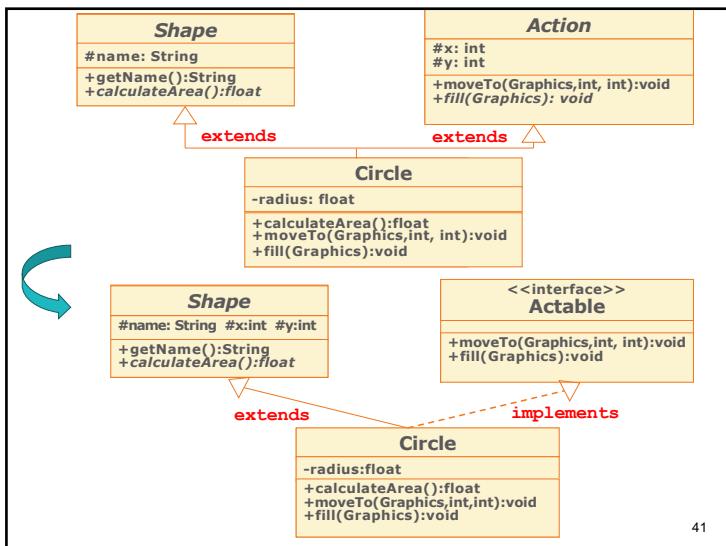


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10



4. Ví dụ giao diện

```

import java.awt.Graphics;
abstract class Shape {
    protected String name;
    protected int x, y;
    Shape(String n, int x, int y) {
        name = n; this.x = x; this.y = y;
    }
    public String getName() {
        return name;
    }
    public abstract float calculateArea();
}

interface Actable {
    public void moveTo(Graphics g, int x1, int y1);
    public void fill(Graphics g);
}

```



42

```

class Circle extends Shape implements Actable {
    private int radius;
    public Circle(String n, int x, int y, int r) {
        super(n, x, y); radius = r;
    }
    public float calculateArea() {
        float area = (float) (3.14 * radius * radius);
        return area;
    }
    public void draw(Graphics g) {
        System.out.println("Draw circle at (" +
                           " + x + "," + y + ")");
        g.drawOval(x-radius,y-radius,2*radius,2*radius);
    }
    public void moveTo(Graphics g, int x1, int y1) {
        x = x1; y = y1; draw(g);
    }
    public void fill(Graphics g) {
        System.out.println("Fill circle at (" +
                           " + x + "," + y + ")");
        // paint the region with color...
    }
}

```

43

4. Giao diện

- ❖ Giao diện có thể được sử dụng như một kiểu dữ liệu
- ❖ Các đối tượng gán cho biến tham chiếu giao diện phải thuộc lớp thực thi giao diện

Ví dụ:

```

public interface I {}
public class A implements I {}
public class B {}
A a = new A();
B b = new B();
I i1 = new A(); // ok
I i2 = new B(); // lỗi

```



44

43

4. Giao diện

- ❖ Một interface có thể được coi như một dạng "class" mà:
 - Phương thức và thuộc tính là public không tuồng minh
 - Các thuộc tính là static và final
 - Các phương thức là abstract
- ❖ Không thể thể hiện hóa (instantiate) trực tiếp

4. Giao diện

- ❖ Góc nhìn quan niệm
 - Interface không cài đặt bất cứ một phương thức nào nhưng để lại cấu trúc thiết kế trên bất cứ lớp nào sử dụng nó
 - Một interface: 1 contract – trong đó các nhóm phát triển phần mềm thống nhất sản phẩm của họ tương tác với nhau như thế nào, mà không đòi hỏi bất cứ một tri thức về cách thức cài đặt chi tiết
 - Interface: đặc tả cho các bản cài đặt (implementation) khác nhau.
 - Phân chia ranh giới:
 - Cái gì (What) và như thế nào (How)
 - Đặc tả và Cài đặt cụ thể.

4. Giao diện

Lớp trừu tượng

- ❖ Cần có ít nhất một phương thức abstract, có thể chứa các phương thức instance
- ❖ Có thể chứa các phương thức protected và static
- ❖ Có thể chứa các thuộc tính final và non-final
- ❖ Một lớp chỉ có thể kế thừa một lớp trừu tượng

Giao diện

- ❖ Chỉ có thể chứa chữ ký phương thức (danh sách các phương thức)
- ❖ Chỉ có thể chứa các phương thức public mà không có mã nguồn
- ❖ Chỉ có thể chứa các thuộc tính hằng
- ❖ Một lớp có thể thực thi (kế thừa) nhiều giao diện

4. Giao diện

Nhược điểm

- Không cung cấp một cách tự nhiên cho các tình huống không có sự dung độ về kế thừa xảy ra
- Kế thừa nhằm tăng tái sử dụng mã nguồn nhưng giao diện không làm được điều này

Bài tập 4

- ❖ 1. Khai báo nào là hợp lệ trong một interface?
`a.public static int answer = 42;`
`b.int answer;`
`c.final static int answer = 42;`
`d.public int answer = 42;`
`e.private final static int answer = 42;`
- ❖ 2. Một lớp có thể kế thừa chính bản thân nó không?
- ❖ 3. Chuyện gì xảy ra nếu lớp cha và lớp con đều có thuộc tính trùng tên?
- ❖ 4. Phát biểu “Các phương thức khởi tạo cũng được thừa kế xuống các lớp con” là đúng hay sai?
- ❖ 5. Có thể xây dựng các phương thức khởi tạo cho lớp trừu tượng không?
- ❖ 6. Có thể khai báo phương thức protected trong một giao diện không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

49

5. Lớp trừu tượng & Giao diện

- ❖ Khi nào nên cho một lớp là lớp độc lập, lớp con, lớp trừu tượng, hay nên biến nó thành interface?
 - Một lớp nên là lớp độc lập, nghĩa là nó không thừa kế lớp nào (ngoại trừ Object) nếu nó không thỏa mãn quan hệ IS-A đối với bất cứ loại nào khác
 - Một lớp nên là lớp con nếu cần cho nó làm một phiên bản chuyên biệt hơn của một lớp khác và cần ghi đè hành vi có sẵn hoặc bổ sung hành vi mới



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

51

Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. **Vai trò của lớp trừu tượng và giao diện**
6. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

50

5. Lớp trừu tượng & Giao diện

- ❖ Khi nào nên cho một lớp là lớp độc lập, lớp con, lớp trừu tượng, hay nên biến nó thành interface?
 - Một lớp nên là lớp cha nếu muốn định nghĩa một khuôn mẫu cho một nhóm các lớp con, và có mã cài đặt mà tất cả các lớp con kia có thể sử dụng
 - Cho lớp đó làm lớp trừu tượng nếu muốn đảm bảo rằng không ai được tạo đối tượng thuộc lớp đó
 - Dùng một interface nếu muốn định nghĩa một vai trò mà các lớp khác có thể nhận, bất kể các lớp đó thuộc cây thừa kế nào



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

52

13

Mở rộng

- Khái niệm giao diện trong các phiên bản của Java

JAVA 7

- Chữ ký các phương thức (method signature)
- Các thuộc tính khai báo hằng (static & final)

JAVA 8

- Chữ ký các phương thức (method signature)
- Các thuộc tính khai báo hằng (static & final)
- Phương thức mặc định (default method)
- Phương thức tĩnh (Static method)
- Private methods

JAVA 9

- Chữ ký các phương thức (method signature)
- Các thuộc tính khai báo hằng (static & final)
- Phương thức mặc định (default method)
- Phương thức tĩnh (Static method)
- Private methods



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

Đa thừa kế

```
interface Interface1 {  
    default void doSomething() {  
        System.out.println("doSomething1");  
    }  
  
interface Interface2 {  
    default void doSomething() {  
        System.out.println("doSomething2");  
    }  
  
public class Multilnheritance implements Interface1, Interface2 {  
    @Override  
    public void doSomething() {  
        Interface1.super.doSomething();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

Ví dụ Java 8 Interface – default methods

<https://gpcoder.com/3854-interface-trong-java-8-default-method-va-static-method/>

```
public interface Shape {  
  
    void draw();  
  
    default void setColor(String color) {  
        System.out.println("Draw shape with color " + color);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

Đa thừa kế

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
  
class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
  
public class Multilnheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        Multilnheritance2 m = new Multilnheritance2();  
        m.doSomething(); // Execute in Parent  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

55

14

Ví dụ Java 8 interface – Static methods

```
interface Vehicle {  
    default void print() {  
        if (isValid())  
            System.out.println("Vehicle printed");  
    }  
    static boolean isValid() {  
        System.out.println("Vehicle is valid");  
        return true;  
    }  
    void showLog();  
}  
  
public class Car implements Vehicle {  
    @Override  
    public void showLog() {  
        print();  
        Vehicle.isValid();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
public class MultiInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultiInheritance2 m = new MultiInheritance2();  
        m.doSomething(); // Execute in Parent  
        m.test(); // OK  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
public class MultiInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultiInheritance2 m = new MultiInheritance2();  
        m.test(); // OK  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
public class MultiInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultiInheritance2 m = new MultiInheritance2();  
        MultiInheritance2.test(); // OK  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
}  
  
public class MultiInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultiInheritance2 m = new MultiInheritance2();  
  
        m.test(); // ERROR!!!  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

61

Tổng kết

❖ Ghi đè

- Các phương thức ở lớp con có cùng chữ ký và danh sách tham số với phương thức ở lớp cha, được tạo ra để định nghĩa lại các hành vi ở lớp con

❖ Lớp trừu tượng

- Các lớp không được khởi tạo đối tượng, được tạo ra làm lớp cơ sở cho các lớp con định nghĩa rõ hơn
- Có ít nhất một phương thức trừu tượng

❖ Giao diện

- Định nghĩa các phương thức mà lớp thực thi phải cài đặt
- Giải quyết vấn đề đa kế thừa



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

63

Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
}  
  
public class MultiInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultiInheritance2 m = new MultiInheritance2();  
  
        Multilnheritance2.test(); // ERROR!!!  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

62

Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

64

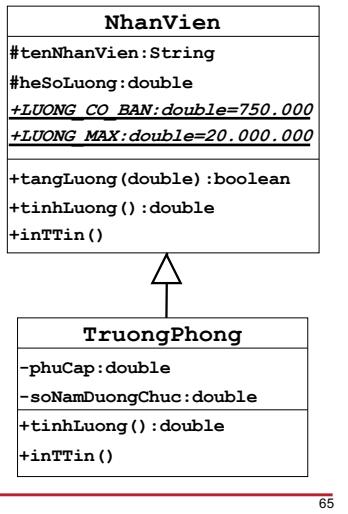
64

Bài tập 5

- ❖ Sửa lại lớp NhanVien:
 - 3 thuộc tính không hằng của NhanVien kế thừa lại cho lớp TruongPhong

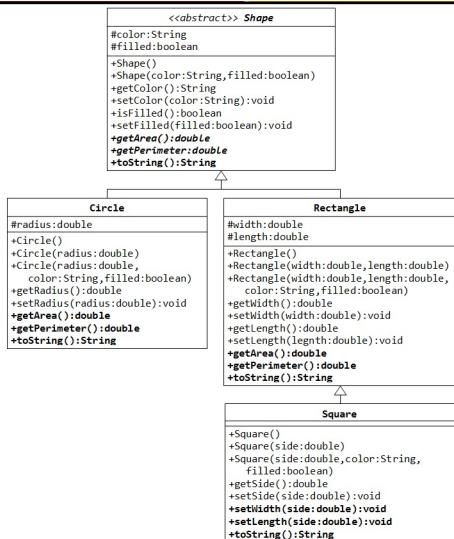
- ❖ Viết mã nguồn của lớp TruongPhong như hình vẽ

- Viết các phương thức khởi tạo cần thiết để khởi tạo các thuộc tính của lớp TruongPhong
- Lương của trưởng phòng = Lương Cơ bản * hệ số lương + phụ cấp



Bài tập 7

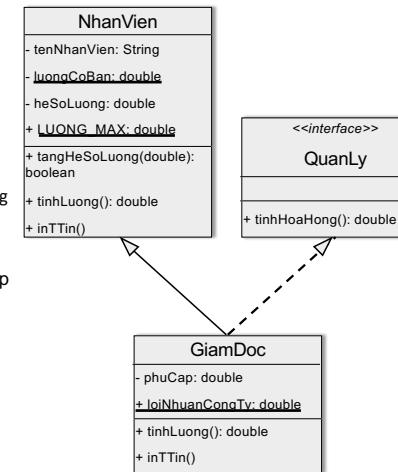
- ❖ Xây dựng các lớp theo sơ đồ lớp



Bài tập 6

- ❖ Viết lớp GiamDoc cài đặt giao diện QuanLy

- Hoa hồng của giám đốc được tính bằng 5% lợi nhuận công ty
- Lương giám đốc = Lương cơ bản * hệ số lương + phụ cấp + hoa hồng
- Tương tự, xây dựng lớp CanBoQuanLy kế thừa lớp NhanVien và thực thi giao diện QuanLy
- Hoa hồng được tính bằng 0.2% lợi nhuận công ty
- CanBoQuanLy không có phụ cấp





ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài 8: Đa hình

1

Mục tiêu

- ❖ Giới thiệu về upcasting và downcasting
- ❖ Phân biệt liên kết tĩnh và liên kết động
- ❖ Nắm vững kỹ thuật đa hình
- ❖ Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

2

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymorphism)
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

3

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymorphism)
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

4

3

4

1. Upcasting và Downcasting

- ❖ Chuyển đổi kiểu dữ liệu nguyên thủy
 - Java tự động chuyển đổi kiểu khi
 - Kiểu dữ liệu tương thích
 - Chuyển đổi từ kiểu hẹp hơn sang kiểu rộng hơn
 - Phải ép kiểu khi
 - Kiểu dữ liệu không tương thích
 - Chuyển đổi từ kiểu rộng hơn sang kiểu hẹp hơn

```
int i;  
double d = i;  
  
byte b; // byte b = (byte)i;
```



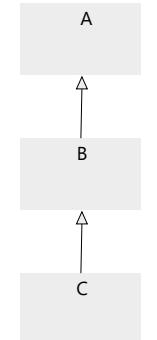
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1. Upcasting và Downcasting

- ❖ Chuyển đổi kiểu dữ liệu tham chiếu
 - Kiểu dữ liệu tham chiếu (lớp) *tương thích*
 - Nằm trên cùng một cây phân cấp kế thừa

```
A var1 = new B();  
  
A var1 = new A();  
C var2 = (C)var1;
```



- Hai loại chuyển đổi
 - Up-casting
 - Down-casting



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

1.1 Upcasting

- ❖ Upcasting: đi lên trên cây phân cấp thừa kế (moving up the inheritance hierarchy)
- ❖ Upcasting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở.
- ❖ Tự động chuyển đổi kiểu

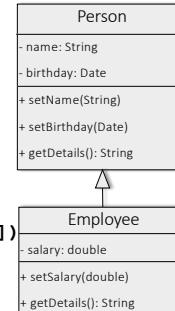


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

Ví dụ Upcasting

```
public class Test1 {  
    public static void main(String arg[]) {  
        Employee e = new Employee();  
        Person p;  
        p = e;  
        p.setName("Hoa");  
        p.setSalary(350000); // compile error  
    }  
}
```

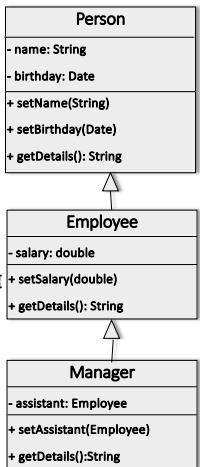


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

Ví dụ Upcasting

```
class Manager extends Employee {  
    Employee assistant;  
    // ...  
    public void setAssistant(Employee e) {  
        assistant = e;  
    }  
    // ...  
}  
  
public class Test2 {  
    public static void main(String arg[]) {  
        Manager junior, senior;  
        // ...  
        senior.setAssistant(junior);  
    }  
}
```

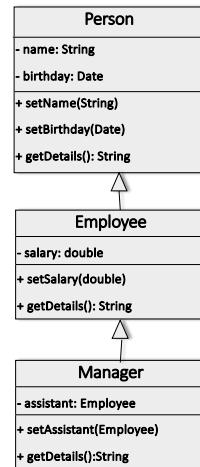


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

Ví dụ Upcasting

```
public class Test3 {  
    String static teamInfo(Person p1,  
                           Person p2) {  
        return "Leader: " + p1.getName() +  
               ", member: " + p2.getName();  
    }  
    public static void main(String arg[]) {  
        Employee e1, e2;  
        Manager m1, m2;  
        // ...  
        System.out.println(teamInfo(e1, e2));  
        System.out.println(teamInfo(m1, m2));  
        System.out.println(teamInfo(m1, e2));  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

1.2 Downcasting

- ❖ Down casting: đi xuống cây phân cấp thừa kế (move back down the inheritance hierarchy)
- ❖ Down casting là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- ❖ Không tự động chuyển đổi kiểu
→ Phải ép kiểu.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

Ví dụ Downcasting

```
public class Test2 {  
    public static void main(String arg[]) {  
        Employee e = new Employee();  
        Person p = e; // upcasting  
        Employee ee = (Employee) p; // downcasting  
        Manager m = (Manager) ee; // run-time error  
  
        Person p2 = new Manager();  
        Employee e2 = (Employee) p2;  
  
        Person p3 = new Employee();  
        Manager e3 = (Manager) p3;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

Toán tử instanceof

- ❖ Kiểm tra xem một đối tượng có phải là thể hiện của một lớp nào đó không
- ❖ Trả về: true | false (nếu đối tượng là null thì trả về false)

```
public class Employee extends Person {}  
public class Student extends Person {}  
  
public class Test{  
    public doSomething(Person e) {  
        if (e instanceof Employee) {...}  
        } else if (e instanceof Student) {...}  
        } else {...}  
    }  
}
```



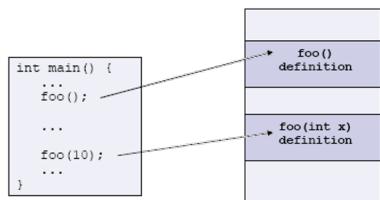
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Liên kết lời gọi hàm

- ❖ Liên kết lời gọi hàm (function call binding) là quy trình xác định khối mã hàm cần chạy khi một lời gọi hàm được thực hiện
 - Ví dụ xử lý liên kết lời gọi hàm trong C: đơn giản vì mỗi hàm có duy nhất một tên



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymorphism)
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Trong ngôn ngữ Hướng đối tượng

- ❖ Liên kết lời gọi phương thức (method call binding): quá trình liên kết lời gọi phương thức tới đoạn code thực thi phương thức
- ❖ Có 2 loại:
 - Liên kết tĩnh (static binding)
 - Liên kết động (dynamic binding)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

2.1 Liên kết tĩnh

- ❖ Liên kết tại thời điểm biên dịch
 - Early Binding/Compile-time Binding
 - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện
 - Nếu có lỗi thì sẽ có lỗi biên dịch
 - Ưu điểm về tốc độ
- ❖ Ví dụ trong Java: các phương thức static



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

2.2 Liên kết động

- ❖ Lời gọi phương thức được quyết định khi thực hiện (run-time)
 - Late binding/Run-time binding
 - Phiên bản của phương thức phù hợp với đối tượng được gọi
 - Java trì hoãn liên kết phương thức cho đến thời gian chạy (run-time) - đây được gọi là liên kết động hoặc liên kết trễ
 - Java mặc định sử dụng liên kết động



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

Ví dụ liên kết tĩnh trong Java

```
class Human{  
    public static void walk() {  
        System.out.println("Human walks");  
    }  
}  
public class Boy extends Human {  
    public static void walk() {  
        System.out.println("Boy walks");  
    }  
}  
public static void main(String args[]) {  
    // Reference is of Human type and object is Boy type  
    Human obj1 = new Boy();  
  
    // Reference is of Human type and object is Human type.  
    Human obj2 = new Human();  
  
    // Reference is of Human type and object is Human type.  
    Boy obj3 = new Boy();  
  
    obj1.walk();  
    obj2.walk();  
    obj3.walk();  
  
    obj1 = obj3;  
    obj1.walk();  
}
```

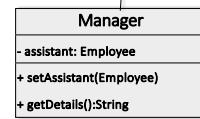
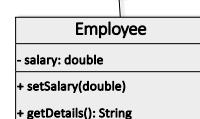
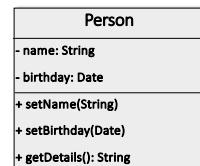


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

Ví dụ

```
public class Test {  
    public static void main(String arg[]) {  
        Person p = new Person();  
        // ...  
        Employee e = new Employee();  
        // ...  
        Manager m = new Manager();  
        // ...  
        Person pArr[] = {p, e, m};  
        for (int i=0; i<pArr.length; i++) {  
            System.out.println(  
                pArr[i].getDetail());  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

Bài tập 1



- ❖ Giả sử lớp Sub kế thừa từ lớp cha Sandwich. Tạo hai đối tượng từ các lớp này:

```
Sandwich x = new Sandwich();  
Sub y = new Sub();
```

- ❖ Phép gán nào sau đây là hợp lệ?

1. x = y;
2. y = x;
3. y = new Sandwich();
4. x = new Sub();

3. Đa hình (Polymorphism)

- ❖ Ví dụ: Nếu đi du lịch, bạn có thể chọn ô tô, thuyền, hoặc máy bay
 - Dù đi bằng phương tiện gì, kết quả cũng giống nhau là bạn đến được nơi cần đến
 - Cách thức đáp ứng các dịch vụ có thể khác nhau



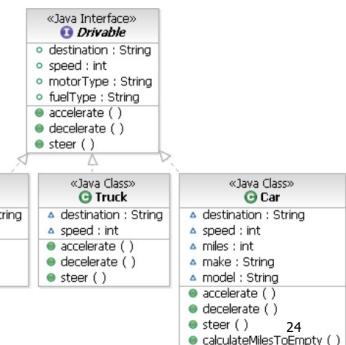
Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. **Đa hình (Polymorphism)**
4. Ví dụ và bài tập

3. Đa hình

- ❖ Các lớp khác nhau có thể đáp ứng danh sách các thông điệp giống nhau, vì vậy cung cấp các dịch vụ giống nhau

- Cách thức đáp ứng thông điệp, thực hiện dịch vụ khác nhau
- Chúng có thể tráo đổi cho nhau mà không ảnh hưởng đến đối tượng gửi thông điệp
- → Đa hình



3. Đa hình

- ❖ Polymorphism: Nhiều hình thức thực hiện, nhiều kiểu tồn tại
 - Khả năng của một biến tham chiếu thay đổi hành vi theo đối tượng mà nó đang tham chiếu tới
- ❖ Đa hình trong lập trình
 - Đa hình phương thức:
 - Phương thức trùng tên, phân biệt bởi danh sách tham số.
 - Đa hình đối tượng
 - Nhận diện đối tượng theo nhiều kiểu khác nhau
 - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

3. Đa hình

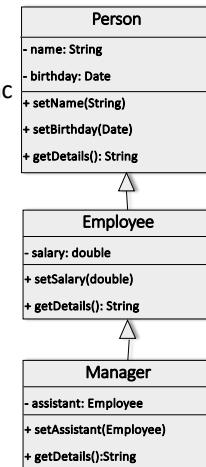
- ❖ Nhận diện đối tượng theo nhiều kiểu khác nhau → Upcasting và Downcasting

```
public class Test3 {  
    public static void main(String args[]){  
        Person p1 = new Employee();  
        Person p2 = new Manager();  
  
        Employee e = (Employee) p1;  
        Manager m = (Manager) p2;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26



3. Đa hình

- ❖ Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau → Liên kết động
- ❖ Ví dụ:

```
Person p1 = new Person();  
Person p2 = new Employee();  
Person p3 = new Manager();  
// ...  
System.out.println(p1.getDetail());  
System.out.println(p2.getDetail());  
System.out.println(p3.getDetail());
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

3. Đa hình

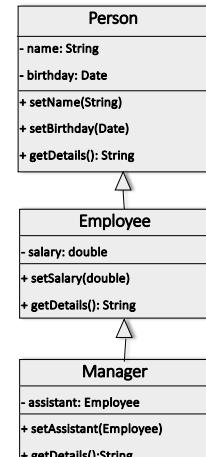
- ❖ Ví dụ:
- ❖ Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau

```
Person p1 = new Person();  
Person p2 = new Employee();  
Person p3 = new Manager();  
// ...  
System.out.println(p1.getDetail());  
System.out.println(p2.getDetail());  
System.out.println(p3.getDetail());
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28



27

7

Toán tử instanceof

```
public class Employee extends Person {}  
public class Student extends Person {}  
  
public class Test{  
    public doSomething(Person e) {  
        if (e instanceof Employee) {...}  
        } else if (e instanceof Student) {...}  
    }  
    } else {...}  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymorphism)
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

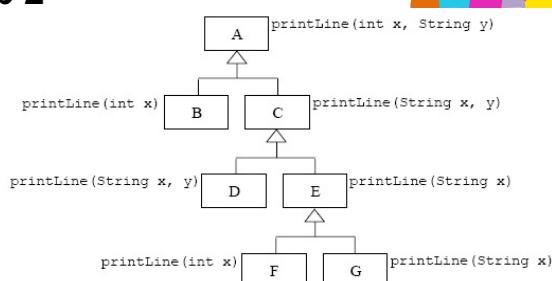
32

Bài tập 2

QUIZ

❖ Cho

biểu đồ lớp:



Phương thức `printLine()` của lớp nào sẽ được sử dụng trong mỗi trường hợp dưới đây, biết rằng `z` là một đối tượng của lớp `F`? Giải thích ngắn gọn?

1. `z.printLine(1)`
2. `z.printLine(2, "Object-Oriented Programming")`
3. `z.printLine("Java")`
4. `z.printLine("Object-Oriented Programming", "Java")`
5. `z.printLine("Object-Oriented Programming", 3)`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

Bài tập 3

**QUIZ
TIME!**

❖ Những điều kiện nào trả về `true`? (Có thể xem Java documentation để biết các quan hệ thừa kế giữa các lớp)
Biết rằng `System.out` là một đối tượng của lớp `PrintStream`.

1. `System.out instanceof PrintStream`
2. `System.out instanceof OutputStream`
3. `System.out instanceof LogStream`
4. `System.out instanceof Object`
5. `System.out instanceof String`
6. `System.out instanceof Writer`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

33

Bài tập 4

- ❖ Kiểm tra các đoạn mã sau đây và vẽ sơ đồ lớp tương ứng

```
abstract public class Animal {  
    abstract public void greeting();  
}  
  
public class Cat extends Animal {  
    public void greeting() {  
        System.out.println("Meow!");  
    }  
}  
  
public class Dog extends Animal {  
    public void greeting() {  
        System.out.println("Woof!");  
    }  
}  
  
public void greeting(Dog another) {  
    System.out.println("Woooooooooof!");  
}  
  
public class BigDog extends Dog {  
    public void greeting() {  
        System.out.println("Woow!");  
    }  
}  
  
public void greeting(Dog another) {  
    System.out.println("Woocooooowwww!");  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

Bài tập 5

- ❖ Giải thích các đầu ra (hoặc các lỗi nếu có) cho chương trình thử nghiệm sau:

```
public class TestAnimal {  
    public static void main(String[] args) {  
        // Using the subclasses  
        Cat cat1 = new Cat();  
        cat1.greeting();  
        Dog dog1 = new Dog();  
        dog1.greeting();  
        BigDog bigDog1 = new BigDog();  
        bigDog1.greeting();  
  
        // Using Polymorphism  
        Animal animal1 = new Cat();  
        animal1.greeting();  
        Animal animal2 = new Dog();  
        animal2.greeting();  
        Animal animal3 = new BigDog();  
  
        animals.greeting();  
        Animal animal4 = new Animal();  
  
        // Downcast  
        Dog dog2 = (Dog)animal2;  
        BigDog bigDog2 = (BigDog)animal3;  
        Dog dog3 = (Dog)animal3;  
        Cat cat2 = (Cat)animal2;  
        dog2.greeting(dog3);  
        dog3.greeting(dog2);  
        dog2.greeting(bigDog2);  
        bigDog2.greeting(dog2);  
        bigDog2.greeting(bigDog1);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

Bài tập 6

- ❖ Phân tích xây dựng các lớp như mô tả sau:
- Hàng điện máy <mã hàng, tên hàng, nhà sản xuất, giá, thời gian bảo hành, điện áp, công suất>
 - Hàng sành sứ <mã hàng, tên hàng, nhà sản xuất, giá, loại nguyên liệu>
 - Hàng thực phẩm <mã hàng, tên hàng, nhà sản xuất, giá, ngày sản xuất, ngày hết hạn dùng>
- ❖ Viết chương trình tạo mỗi loại một mặt hàng cụ thể. Xuất thông tin về các mặt hàng này.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

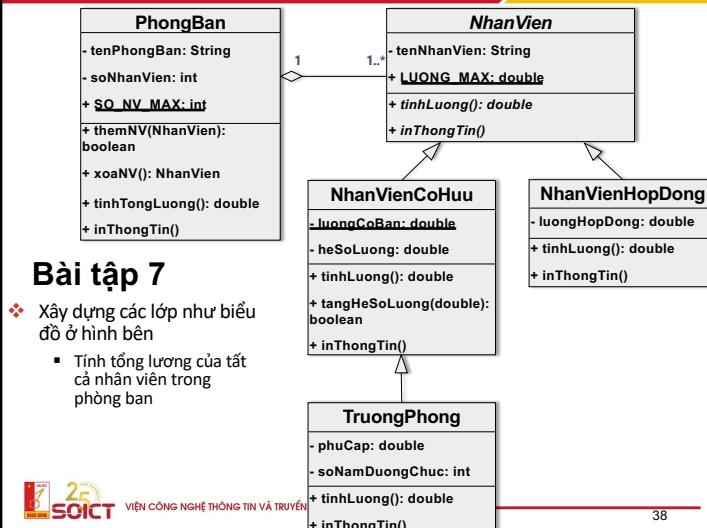
Bài tập 7

- ❖ Xây dựng các lớp như biểu đồ ở hình bên
- Tính tổng lương của tất cả nhân viên trong phòng ban



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38





40

40

10



Bài 9: Lập trình tổng quát

1

Mục tiêu

- ❖ Giới thiệu về lập trình tổng quát và cách thực hiện trong các ngôn ngữ lập trình
- ❖ Giới thiệu về collection framework với các cấu trúc tổng quát: List, HashMap, Tree, Set, Vector,...
- ❖ Định nghĩa và sử dụng Template và ký tự đại diện (wildcard)
- ❖ Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



2

Nội dung

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập



3

Nội dung

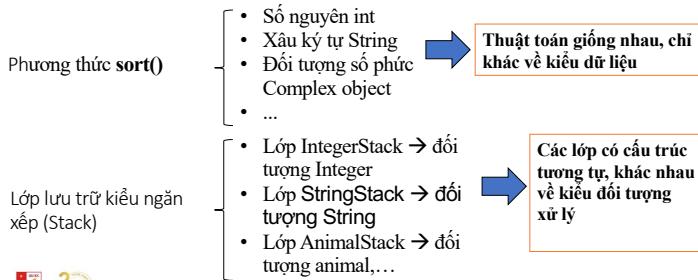
1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập



4

1. Giới thiệu về lập trình tổng quát

- ❖ Lập trình tổng quát (Generic programming): Tổng quát hóa chương trình để có thể hoạt động với các kiểu dữ liệu khác nhau, kể cả kiểu dữ liệu trong tương lai
 - Thuật toán đã xác định
- ❖ Ví dụ:



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

5

1. Giới thiệu về lập trình tổng quát

- ❖ Ví dụ C: hàm `memcpy()` trong thư viện `string.h`

```
void* memcpy(void* region1, const void* region2, size_t n);
```

 - Hàm `memcpy()` bên trên được khai báo tổng quát bằng cách sử dụng các con trỏ `void*`
 - Điều này giúp cho hàm có thể sử dụng với nhiều kiểu dữ liệu khác nhau
 - Dữ liệu được truyền vào một cách tổng quát thông qua địa chỉ và kích thước kiểu dữ liệu
 - Hay nói cách khác, để sao chép dữ liệu, ta chỉ cần địa chỉ và kích cỡ của chúng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

7

1. Giới thiệu về lập trình tổng quát

- ❖ Lập trình tổng quát
 - C: dùng con trỏ không định kiểu (con trỏ `void`)
 - C++: dùng template
 - Java 1.5 trở về trước: lợi dụng upcasting, downcasting và lớp object
 - Java 1.5: đưa ra khái niệm về template



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

6

1. Giới thiệu về lập trình tổng quát

- ❖ Ví dụ: Lập trình tổng quát từ trước Java 1.5

```
public class ArrayList {  
    public Object get(int i) { . . . }  
    public void add(Object o) { . . . }  
  
    private Object[] elementData;  
}
```
- ❖ Lớp Object là lớp cha tổng quát nhất → có thể chấp nhận các đối tượng thuộc lớp con của nó

```
List myList = new ArrayList();  
myList.add("Fred");  
myList.add(new Dog());  
myList.add(new Integer(42));
```

Các đối tượng trong một danh sách khác hẳn nhau
- ❖ Hạn chế: Phải ép kiểu → có thể ép sai kiểu (run-time error)

```
String name = (String) myList.get(1); //Dog!!!
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

2

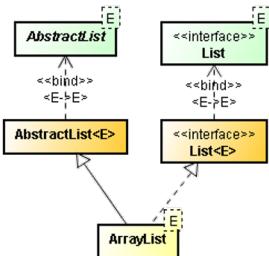
1. Giới thiệu về lập trình tổng quát

- Ví dụ: Lập trình Generic từ Java 1.5

- Java 1.5 Template

Danh sách chỉ chấp nhận các đối tượng có kiểu là Integer

```
List<Integer> myList =  
    new LinkedList<Integer>();  
myList.add(new Integer(0));  
Integer x = myList.iterator().next(); //Không cần ép kiểu  
  
myList.add(new String("Hello")); //Compile Error
```



9

9

Lớp tổng quát

- Lớp tổng quát (generic class) là lớp có thể nhận kiểu dữ liệu là một lớp bất kỳ
- Cú pháp
 - Tên Lớp <kiểu 1, kiểu 2, kiểu 3...>
 - {
 - }
- Các phương thức hay thuộc tính của lớp tổng quát có thể sử dụng các kiểu được khai báo như mọi lớp bình thường khác

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

Nội dung

- Giới thiệu về lập trình tổng quát
- Định nghĩa và sử dụng Template
- Lập trình tổng quát trong Java collections framework
- Ký tự đại diện (Wildcard)
- Ví dụ và bài tập

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

Lớp tổng quát

Tên kiểu, sẽ được thay thế bằng một kiểu cụ thể khi sử dụng

- Ví dụ:

```
public class Information<T> {  
    private T value;  
    public Information(T value) {  
        this.value = value;  
    }  
    public T getValue() {  
        return value;  
    }  
}  
Information<String> mystring =  
    new Information<String>("hello");  
Information<Circle> circle =  
    new Information<Circle>(new Circle());  
Information<2DShape> shape =  
    new Information<>(new 2DShape());
```

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

12

Lớp tổng quát

- ❖ Quy ước đặt tên kiểu

Tên kiểu	Mục đích
E	Các thành phần trong một collection
K	Kiểu khóa trong Map
V	Kiểu giá trị trong Map
T	Các kiểu thông thường
S, U	Các kiểu thông thường khác

- ❖ Chú ý: Không sử dụng các kiểu dữ liệu nguyên thủy cho các lớp tổng quát

```
Information<int> integer =  
    new Information<int>(2012);      // Error  
Information<Integer> integer =  
    new Information<Integer>(2012); // OK
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Ví dụ Phương thức tổng quát

```
public class ArrayTool {  
    // Phương thức in các phần tử trong mảng String  
    public static void print(String[] a) {  
        for (String e : a) System.out.print(e + " ");  
        System.out.println();  
    }  
  
    // Phương thức in các phần tử trong mảng với kiểu  
    // dữ liệu bất kỳ  
    public static <E> void print(E[] a) {  
        for (E e : a) System.out.print(e + " ");  
        System.out.println();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

Phương thức tổng quát

- ❖ Phương thức tổng quát (generic method) là các phương thức tự định nghĩa kiểu tham số của nó
- ❖ Có thể được viết trong lớp bất kỳ (tổng quát hoặc không)
- ❖ Cú pháp
 - (chỉ định truy cập) **<kiểu1, kiểu 2...>**
 - (kiểu trả về) tên phương thức (danh sách tham số) {
 //...
}
- ❖ Ví dụ

```
public static <E> void print(E[] a) { ... }
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

Ví dụ Phương thức tổng quát

```
...  
String[] str = new String[5];  
Point[] p = new Point[3];  
int[] intnum = new int[2];  
  
ArrayTool.print(str);  
ArrayTool.print(p);  
  
// Không dùng được với kiểu dữ liệu nguyên thủy  
ArrayTool.print(intnum);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

Giới hạn kiểu dữ liệu tổng quát

- ❖ Có thể giới hạn các kiểu dữ liệu tổng quát sử dụng phải là dẫn xuất của một hoặc nhiều lớp
- ❖ Giới hạn 1 lớp
`<type_param extends bound>`
- ❖ Giới hạn nhiều lớp
`<type_param extends bound_1 & bound_2 & ...>`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

Nội dung

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

19

Giới hạn kiểu dữ liệu tổng quát

- ❖ Ví dụ:

```
public class Information<T extends 2DShape> {  
    private T value;  
    public Information(T value) {  
        this.value = value;  
    }  
    public T getValue() {  
        return value;  
    }  
}  
Information<Point> pointInfo =  
    new Information<Point>(new Point()); // OK  
Information<String> stringInfo =  
    new Information<String>(); // error
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

18

3. Java Collections Framework

- ❖ Collection là đối tượng có khả năng chứa các đối tượng khác.
- ❖ Các thao tác thông thường trên collection
 - Thêm/Xoá đối tượng vào/khỏi collection
 - Kiểm tra một đối tượng có ở trong collection không
 - Lấy một đối tượng từ collection
 - Duyệt các đối tượng trong collection
 - Xoá toàn bộ collection



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

20

3. Java Collections Framework

- ❖ Các collection đầu tiên của Java:
 - Mảng
 - Vector: Mảng động
 - Hastable: Bảng băm
- ❖ Collections Framework (từ Java 1.2)
 - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
 - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

3. Java Collections Framework

- ❖ Collections Framework bao gồm
 - Interfaces: Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
 - Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
 - Algorithms: Là các phương thức tinh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

3. Java Collections Framework

- ❖ Một số lợi ích của Collections Framework
 - Giảm thời gian lập trình
 - Tăng cường hiệu năng chương trình
 - Dễ mở rộng các collection mới
 - Khuyến khích việc sử dụng lại mã chương trình



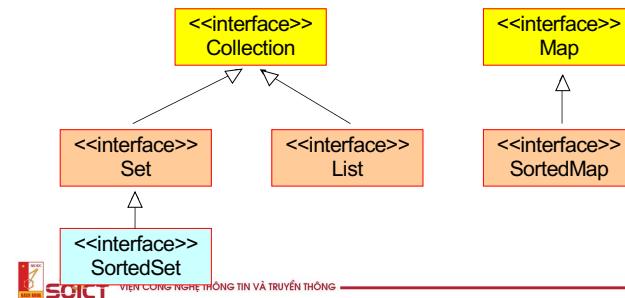
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

Interfaces trong Java collections framework

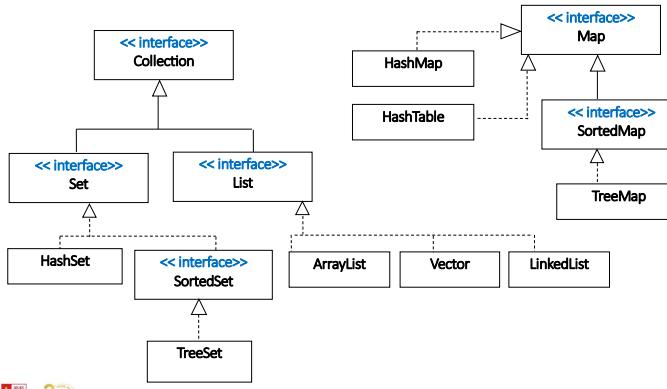
- ❖ List: Tập các đối tượng tuần tự, kế tiếp nhau, có thể lặp lại
- ❖ Set: Tập các đối tượng không lặp lại
- ❖ Map: Tập các cặp khóa-giá trị (key-value) và không cho phép khóa lặp lại



24

24

3. Java Collections Framework



25

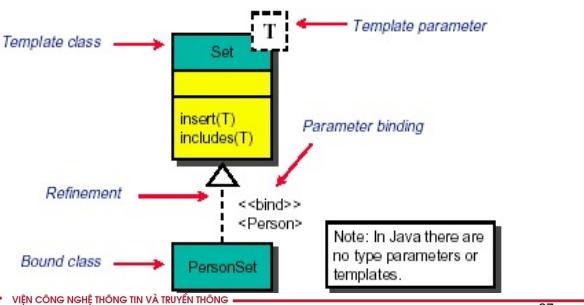
So sánh Collection và Array

Collection	Array
Collection (có thể) truy xuất theo dạng ngẫu nhiên	Mảng truy xuất 1 cách tuần tự
Collection có thể chứa nhiều loại đối tượng/dữ liệu khác nhau	Mảng chứa 1 loại đối tượng/dữ liệu nhất định
Dùng Java Collection, chỉ cần khai báo và gọi những phương thức đã được định nghĩa sẵn	Dùng tổ chức dữ liệu theo mảng phải lập trình hoàn toàn
Duyệt các phần tử tập hợp thông qua Iterator	Duyệt các phần tử mảng tuần tự thông qua chỉ số mảng

26

3. Java Collections Framework

- Các giao diện và lớp thực thi trong Collection framework của Java đều được xây dựng theo template
 - Cho phép xác định tập các phần tử cùng kiểu nào đó bất kỳ
 - Cho phép chỉ định kiểu dữ liệu của các Collection → hạn chế việc thao tác sai kiểu dữ liệu



27

Ví dụ

```

public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}

List<String> myList = new ArrayList<String>();
myList.add("Fred");           // OK
myList.add(new Dog());        // Compile error!

String s = myList.get(0);
  
```

28

Giao diện Collection

- ❖ Xác định giao diện cơ bản cho các thao tác với một tập các đối tượng
 - Thêm vào collection
 - Xóa khỏi collection
 - Kiểm tra có là thành viên
- ❖ Chứa các phương thức thao tác trên các phần tử riêng lẻ hoặc theo khối
- ❖ Cung cấp các phương thức cho phép thực hiện duyệt qua các phần tử trên collection (lặp) và chuyển tập hợp sang mảng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

29

Giao diện Collection

```
public interface Collection {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(Object element);  
    boolean remove(Object element);  
    Iterator iterator();  
  
    // Bulk Operations  
    boolean addAll(Collection c);  
    boolean removeAll(Collection c);  
    boolean retainAll(Collection c);  
  
    ...  
    // Array Operations  
    Object[] toArray();  
    Object[] toArray(Object a[]);  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

30

Giao diện Set

- ❖ Set kế thừa từ Collection nên cũng hỗ trợ toàn bộ các thao tác xử lý trên Collection

Ví dụ:

- Set of cars:
 - {BMW, Ford, Jeep, Chevrolet, Nissan, Toyota, VW}
- Nationalities in the class
 - {Chinese, American, Canadian, Indian}

- ❖ Set là một tập hợp các phần tử **không được trùng lặp**.

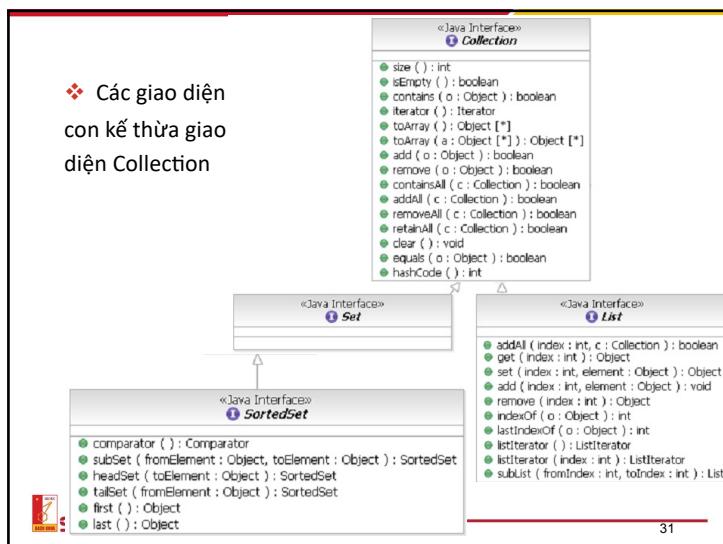
- ❖ Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

32



31

Giao diện SortedSet

- ❖ **SortedSet:** kế thừa giao diện Set
 - Các phần tử được sắp xếp theo một thứ tự
 - Không có các phần tử trùng nhau
 - Cho phép một phần tử là null
 - Các đối tượng đưa vào trong một SortedSet phải cài đặt giao diện Comparable hoặc lớp cài đặt SortedSet phải nhận một Comparator trên kiểu của đối tượng đó
- ❖ **Một số phương thức:**
 - first(): lấy phần tử đầu tiên (nhỏ nhất)
 - last(): lấy phần tử cuối cùng (lớn nhất)
 - SortedSet subSet(Object e1, Object e2): lấy một tập các phần tử nằm trong khoảng từ e1 tới e2

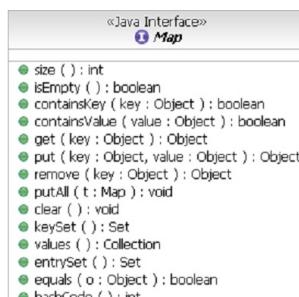


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

Giao diện Map

- ❖ Xác định giao diện cơ bản để thao tác với một tập hợp bao gồm cặp khóa-giá trị
 - Thêm một cặp khóa-giá trị
 - Xóa một cặp khóa-giá trị
 - Lấy về giá trị với khóa đã có
 - Kiểm tra có phải là thành viên (khóa hoặc giá trị)
- ❖ Cung cấp 3 cách nhìn cho nội dung của tập hợp:
 - Tập các khóa
 - Tập các giá trị
 - Tập các ánh xạ khóa-giá trị



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

Giao diện List

- ❖ List kế thừa từ Collection. List cung cấp thêm các phương thức để xử lý Collection kiểu danh sách
 - Danh sách là một collection với các phần tử được xếp theo chỉ số
- ❖ **Một số phương thức của List**
 - Object get(int index);
 - Object set(int index, Object o);
 - void add(int index, Object o);
 - Object remove(int index);
 - int indexOf(Object o);
 - int lastIndexOf(Object o);



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

Giao diện Map

- ❖ Giao diện Map cung cấp các thao tác xử lý trên các bảng ánh xạ
 - Bảng ánh xạ lưu các phần tử theo khóa và không được có 2 khóa trùng nhau
- ❖ **Một số phương thức của Map**
 - Object put(Object key, Object value);
 - Object get(Object key);
 - Object remove(Object key);
 - boolean containsKey(Object key);
 - boolean containsValue(Object value);
 - ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

Giao diện SortedMap

❖ Giao diện SortedMap

- thừa kế giao diện Map
- các phần tử được sắp xếp theo thứ tự
- tương tự SortedSet, tuy nhiên việc sắp xếp được thực hiện với các khóa

❖ Phương thức: Tương tự Map, bổ sung thêm:

- `firstKey()`: returns the first (lowest) value currently in the map
- `lastKey()`: returns the last (highest) value currently in the map



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

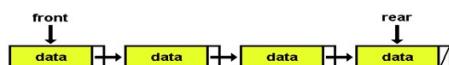
37

Các lớp thực thi giao diện Collection

❖ ArrayList: Mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ

❖ LinkedList: Danh sách liên kết

- Hỗ trợ thao tác trên đầu và cuối danh sách
- Được sử dụng để tạo ngăn xếp, hàng đợi, cây...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

Các lớp thực thi giao diện Collection

- ❖ Java đã xây dựng sẵn một số lớp thực thi các giao diện Set, List và Map và cài đặt các phương thức tương ứng

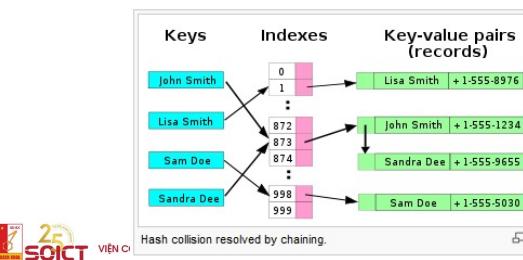
		IMPLEMENTATIONS				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Legacy
INTERFACES	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		Hashtable, Properties

38

Các lớp thực thi giao diện Collection

❖ HashSet: Bảng băm

- Lưu các phần tử trong một bảng băm
- Không cho phép lưu trùng lặp
- Cho phép phần tử null



40

10

Các lớp thực thi giao diện Collection

- ❖ **LinkedHashSet**: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử
 - Thừa kế HashSet và thực thi giao diện Set
 - Khác HashSet ở chỗ nó lưu trữ trong một danh sách mốc nối đôi
 - Thứ tự các phần tử được sắp xếp theo thứ tự được insert vào tập hợp
- ❖ **TreeSet**: Cho phép lấy các phần tử trong tập hợp theo thứ tự đã sắp xếp
 - Các phần tử được thêm vào TreeSet tự động được sắp xếp
 - Thông thường, ta có thể thêm các phần tử vào HashSet, sau đó convert về TreeSet để duyệt theo thứ tự nhanh hơn



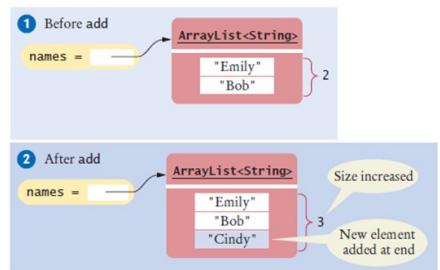
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

Ví dụ

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Emily");  
names.add("Bob");  
names.add("Cindy");
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

Các lớp thực thi giao diện Collection

- ❖ **HashMap**: Bảng băm (cài đặt của Map)
- ❖ **LinkedHashMap**: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử (cài đặt của Map)
- ❖ **TreeMap**: Cây (cài đặt của Map)
- ❖ **Legacy Implementations**
 - Là các lớp cũ được cài đặt bổ sung thêm các collection interface.
 - **Vector**: Có thể thay bằng ArrayList
 - **Hashtable**: Có thể thay bằng HashMap



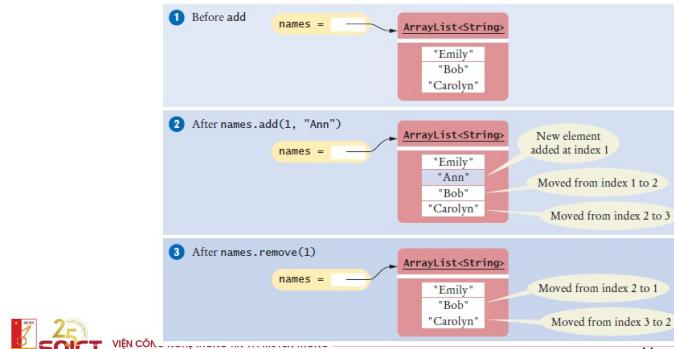
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

Ví dụ

```
String name = names.get(0);  
names.add(1, "Ann");  
names.remove(1);
```



VIỆN CÔNG NGHỆ THÔNG TIN

44

44

Bài tập 1



- Sau khi thực hiện đoạn chương trình sau, danh sách names có chứa các phần tử nào?

```
ArrayList<String> names = new ArrayList<String>();
names.add("Bob");
names.add(0, "Ann");
names.remove(1);
names.add("Cal");
```



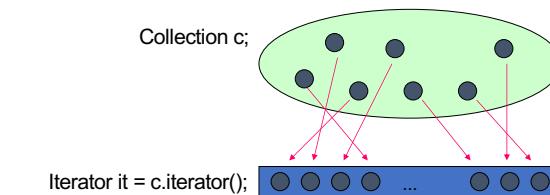
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

45

Giao diện Iterator và Comparator

- Sử dụng để duyệt và so sánh trên các Collection
- Iterator
 - Các phần tử trong collection có thể được duyệt thông qua Iterator



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

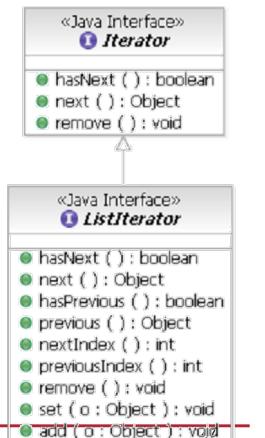
46

46

Giao diện Iterator và Comparator

Iterator

- Cung cấp cơ chế thuận tiện để duyệt (lặp) qua toàn bộ nội dung của tập hợp, mỗi lần là một đối tượng trong tập hợp
 - Giống như SQL cursor
- Iterator của các tập hợp đã sắp xếp duyệt theo thứ tự tập hợp
- ListIterator thêm các phương thức đưa ra bản chất tuần tự của danh sách cơ sở



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

Giao diện Iterator và Comparator

Iterator : Các phương thức

- iterator(): yêu cầu container trả về một iterator
- next(): trả về phần tử tiếp theo
- hasNext(): kiểm tra có tồn tại phần tử tiếp theo hay không
- remove(): xóa phần tử gần nhất của iterator



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

48

Giao diện Iterator và Comparator

- ❖ Iterator: Ví dụ

- Định nghĩa iterator

```
public interface Iterator {  
    boolean hasNext();  
    Object next();  
    void remove();  
}  
  
▪ Sử dụng iterator  
Collection c;  
  
Iterator i = c.iterator();  
while (i.hasNext()) {  
    Object o = i.next();  
    // Process this object  
}
```

Tương tự vòng lặp for
for (String name : names){
 System.out.println(name);
}



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

49

Giao diện Iterator và Comparator

- ❖ Ví dụ lớp Person:

```
class Person {  
    private int age;  
    private String name;  
  
    public void setAge(int age){  
        this.age=age;  
    }  
    public int getAge(){  
        return this.age;  
    }  
    public void setName(String name){  
        this.name=name;  
    }  
    public String getName(){  
        return this.name;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

51

Giao diện Iterator và Comparator

- ❖ Giao diện Comparator được sử dụng để so sánh hai đối tượng trong tập hợp

- ❖ Một Comparator phải định nghĩa một phương thức compare() lấy 2 tham số Object và trả về -1, 0 hoặc 1

- ❖ Không cần thiết nếu tập hợp đã có khả năng so sánh tự nhiên (vd. String, Integer...)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

50

Giao diện Iterator và Comparator

- ❖ Ví dụ Cài đặt AgeComparator :

```
class AgeComparator implements Comparator {  
    public int compare(Object ob1, Object ob2) {  
        int ob1Age = ((Person)ob1).getAge();  
        int ob2Age = ((Person)ob2).getAge();  
  
        if(ob1Age > ob2Age)  
            return 1;  
        else if(ob1Age < ob2Age)  
            return -1;  
        else  
            return 0;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

52

13

Giao diện Iterator và Comparator

- ❖ Ví dụ Sử dụng AgeComparator :

```
public class ComparatorExample {  
    public static void main(String args[]) {  
        ArrayList<Person> lst = new  
            ArrayList<Person>();  
        Person p = new Person();  
        p.setAge(35); p.setName("A");  
        lst.add(p);  
  
        p = new Person();  
        p.setAge(30); p.setName("B");  
        lst.add(p);  
  
        p = new Person();  
        p.setAge(32); p.setName("C");  
        lst.add(p);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

Nội dung

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. **Ký tự đại diện (Wildcard)**
5. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

Giao diện Iterator và Comparator

- ❖ Ví dụ Sử dụng AgeComparator :

```
System.out.println("Order before sorting");  
for (Person person : lst) {  
    System.out.println(person.getName() +  
        "\t" + person.getAge());  
}  
  
Collections.sort(lst, new AgeComparator());  
System.out.println("\n\nOrder of person" +  
    "after sorting by age");  
  
for (Iterator<Person> i = lst.iterator();  
    i.hasNext();) {  
    Person person = i.next();  
    System.out.println(person.getName() + "\t" +  
        person.getAge());  
} //End of for  
} //End of main  
} //End of class
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

4. Ký tự đại diện (Wildcard)

- ❖ Quan hệ thừa kế giữa hai lớp không có ảnh hưởng gì đến quan hệ giữa các cấu trúc tổng quát dùng cho hai lớp đó.

- ❖ Ví dụ:

- Dog và Cat là các lớp con của Animal
- Có thể đưa các đối tượng Dog và Cat vào một ArrayList<Animal> (sử dụng phương thức add)
- Tuy nhiên, ArrayList<Dog>, ArrayList<Cat> lại không có quan hệ gì với ArrayList<Animal>



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

4. Ký tự đại diện (Wildcard)

- Không thể ép kiểu ArrayList<Child> về kiểu ArrayList<Parent>

```
class Parent { }
class Child extends Parent { }

ArrayList<Parent> myList = new ArrayList<Child>();
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

4. Ký tự đại diện (Wildcard)

- Giải pháp: sử dụng kí tự đại diện (wildcard)
- Ký tự đại diện: ? dùng để hiển thị cho một kiểu dữ liệu bất kỳ
- Khi biên dịch, dấu ? có thể được thay thế bởi bất kì kiểu dữ liệu nào.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

Ví dụ

```
public class Test {
    public static void main(String args[]) {
        List<String> lst0 = new LinkedList<String>();
        List<Object> lst1 = lst0; // Error
        printList(lst0); // Error
    }

    void static printList(List<Object> lst) {
        Iterator it = lst.iterator();
        while (it.hasNext())
            System.out.println(it.next());
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

Ví dụ: Sử dụng Wildcards

```
public class Test {
    void printList(List<?> lst) {
        Iterator it = lst.iterator();
        while (it.hasNext())
            System.out.println(it.next());
    }

    public static void main(String args[]) {
        List<String> lst0 = new LinkedList<String>();
        List<Employee> lst1 = new LinkedList<Employee>();

        printList(lst0); // String
        printList(lst1); // Employee
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

59

15

4. Ký tự đại diện (Wildcard)

- ❖ Lưu ý: cách làm sau là không hợp lệ

```
ArrayList<?> list = new  
    ArrayList<String>();  
list.add("a1"); //compile error  
list.add(new Object()); //compile error
```

- ❖ Nguyên nhân: Vì không biết list là danh sách liên kết cho kiểu dữ liệu nào, nên không thể thêm phần tử vào list, kể cả đối tượng của lớp Object

4. Ký tự đại diện (Wildcard)

- ❖ Ví dụ:

- `? extends Animal` có nghĩa là kiểu gì đó thuộc loại Animal (là Animal hoặc con của Animal)

- ❖ Lưu ý: Hai cú pháp sau là tương đương:

```
public void foo( ArrayList<? extends Animal> a)  
public <T extends Animal> void foo( ArrayList<T> a)
```

4. Ký tự đại diện (Wildcard)

- ❖ "? extends Type": Xác định một tập các kiểu con của Type. Đây là wildcard hữu ích

- ❖ "? super Type": Xác định một tập các kiểu cha của Type

- ❖ "?": Xác định tập tất cả các kiểu hoặc bất kỳ kiểu nào

Khác biệt giữa print1 và print2?

```
public void print1(List<Employee> list) {  
    for (Employee e : list) {  
        System.out.println(e);  
    }  
}
```

```
public void print2(List<? extends Employee> list) {  
    for (Employee e : list) {  
        System.out.println(e);  
    }  
}
```

Nội dung

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. **Ví dụ và bài tập**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

65

Bài tập 2

Bài tập 2

- ❖ Trưu tượng hoá mô tả sau: một quyển sách là tập hợp các chương, chương là tập hợp các trang.
 - Phác họa các lớp Book, Chapter, và Page
 - Tạo các thuộc tính cần thiết cho các lớp, sử dụng Collection
 - Tạo các phương thức cho lớp Chapter cho việc thêm trang và xác định một chương có bao nhiêu trang
 - Tạo các phương thức cho lớp Book cho việc thêm chương và xác định quyển sách có bao nhiêu chương, và số trang cho quyển sách



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

66

Bài tập 3

- ❖ Xây dựng lớp Stack tổng quát với các kiểu dữ liệu

StackOfChars
- elements: char[]
- size: int
+ StackOfChars()
+ StackOfChars (capacity: int)
+ isEmpty(): boolean
+ isFull(): boolean
+ peak(): char
+ push(value:char): void
+ pop(): char
+ getSize(): int

StackOfIntegers
- elements: int[]
- size: int
+ StackOfIntegers()
+ StackOfIntegers (capacity: int)
+ isEmpty(): boolean
+ isFull(): boolean
+ peak(): int
+ push(value:int): void
+ pop(): int
+ getSize(): int

...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

67

67

68



25

SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



68

17



Bài 10. Ngoại lệ và xử lý ngoại lệ

1

Mục tiêu của bài học

- ❖ Giải thích về ngoại lệ là gì và mô tả các lợi ích của việc xử lý ngoại lệ hướng đối tượng
- ❖ Giải thích được mô hình xử lý ngoại lệ
- ❖ Sử dụng khối try/catch/finally để bắt và xử lý ngoại lệ trong Java
- ❖ Hiểu và biết cách sử dụng ủy nhiệm ngoại lệ
- ❖ Biết cách tạo ra và sử dụng ngoại lệ tự định nghĩa



2

Nội dung

1. Ngoại lệ
2. Bắt và xử lý ngoại lệ
3. Ủy nhiệm ngoại lệ
4. Tạo ngoại lệ tự định nghĩa



3

Nội dung

1. Нgoại lê
2. Bắt và xử lý ngoại lệ
3. Ủy nhiệm ngoại lệ
4. Tạo ngoại lệ tự định nghĩa



4

1.1. Ngoại lệ là gì?

- ❖ Exception = Exceptional event
- ❖ Định nghĩa: Ngoại lệ là một sự kiện xảy ra trong quá trình thực thi chương trình, nó phá vỡ luồng bình thường của chương trình

Ví dụ:



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1.2. Cách xử lý lỗi truyền thông

- ❖ Viết mã xử lý tại nơi phát sinh ra lỗi
 - Làm cho chương trình trở nên rối
 - Không phải lúc nào cũng đầy đủ thông tin để xử lý
 - Không nhất thiết phải xử lý
- ❖ Truyền trạng thái lên mức trên
 - Thông qua tham số, giá trị trả lại hoặc biến tổng thể (flag)
 - Dễ nhầm
 - Vẫn còn khó hiểu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

1.1. Ngoại lệ là gì? (2)

- ❖ Ngoại lệ là một lỗi đặc biệt
- ❖ Khi xảy ra một ngoại lệ, nếu không xử lý thì chương trình kết thúc ngay và trả lại quyền điều khiển cho hệ điều hành.

```
float number1, number2;  
//input number1, number2  
float division = number1 / number2;
```

No handler exists

EXIT



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Ví dụ

```
int devide(int num, int denom, int *error)  
{  
    if (denom != 0){  
        error = 0;  
        return num/denom;  
    } else {  
        error = 1;  
        return 0;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

7

2

Nhược điểm

- ❖ Khó kiểm soát được hết các trường hợp
 - Lỗi số học, lỗi bộ nhớ,...
- ❖ Lập trình viên thường quên không xử lý lỗi
 - Bản chất con người
 - Thiếu kinh nghiệm, cố tình bỏ qua



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

Nội dung

1. Ngoại lệ
2. Bắt và xử lý ngoại lệ
3. Ủy nhiệm ngoại lệ
4. Tạo ngoại lệ tự định nghĩa



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

2.1. Mục đích của xử lý ngoại lệ

- ❖ Giúp chương trình đáng tin cậy hơn, tránh kết thúc bất thường
- ❖ Tách biệt khỏi lệnh có thể gây ngoại lệ và khỏi lệnh xử lý ngoại lệ

```
.....  
IF B IS ZERO GO TO ERROR  
C = A/B  
PRINT C  
GO TO EXIT
```

ERROR:
DISPLAY "DIVISION BY ZERO"

EXIT:
END

} Khối xử lý lỗi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

2.1. Mục đích của xử lý ngoại lệ (2)

- ❖ Khi xảy ra ngoại lệ, nếu không có cơ chế xử lý thích hợp:
 - Chương trình bị ngắt khi ngoại lệ xảy ra
 - Các tài nguyên không được giải phóng → lãng phí
- ❖ Ví dụ: Vào/ra tệp tin
 - Nếu ngoại lệ xảy ra (ví dụ như chuyển đổi kiểu không đúng)
→ Chương trình kết thúc mà không đóng tệp tin lại
 - Tệp tin không thể truy cập/hóng
 - Tài nguyên cấp phát không được giải phóng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

12

2.2. Mô hình xử lý ngoại lệ

❖ Hướng đối tượng

- Đóng gói các điều kiện không mong đợi trong một đối tượng
- Khi xảy ra ngoại lệ, đối tượng tương ứng với ngoại lệ được tạo ra chứa thông tin chi tiết về ngoại lệ
- Cung cấp cơ chế hiệu quả trong việc xử lý lỗi
- Tách biệt luồng điều khiển bất thường với luồng bình thường

```
float sales = getSales();
int staffsize = getStaff().size();
float avg_sales = sales/staffsize;
System.out.println(avg_sales);
```

handler



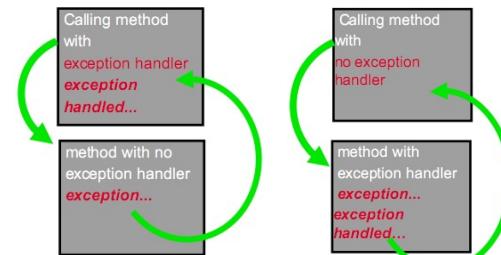
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

2.2. Mô hình xử lý ngoại lệ (2)

- ❖ Ngoại lệ cần phải được xử lý ở tại phương thức sinh ra ngoại lệ hoặc ủy nhiệm cho phương thức gọi đến



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

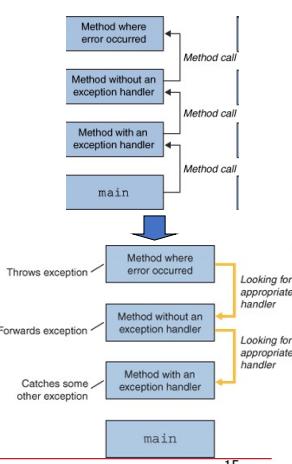
14

2.3. Xử lý ngoại lệ trong Java

- ❖ Java có cơ chế xử lý ngoại lệ rất mạnh

- ❖ Xử lý ngoại lệ trong Java được thực hiện theo mô hình hướng đối tượng:

- Tất cả các ngoại lệ đều là thể hiện của một lớp kế thừa từ lớp Throwable hoặc các lớp con của nó
- Các đối tượng này có nhiệm vụ chuyển thông tin về ngoại lệ (loại và trạng thái của chương trình) từ vị trí xảy ra ngoại lệ đến nơi quản lý/xử lý nó.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

2.3. Xử lý ngoại lệ trong Java (2)

- ❖ Các từ khóa

- try
- catch
- finally
- throw
- throws



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

2.3.1. Khối try/catch

- Khối try ... catch: Phân tách đoạn chương trình thông thường và phần xử lý ngoại lệ
 - try {...}: Khối lệnh có khả năng gây ra ngoại lệ
 - catch() {...}: Bắt và xử lý với ngoại lệ

```
try {  
    // Đoạn ma có thể gây ngoại lệ  
} catch (ExceptionType e) {  
    // Xử lý ngoại lệ  
}  
  
❖ ExceptionType là một lớp con của Throwable
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

Ví dụ có xử lý ngoại lệ

```
class ArgExceptionDemo {  
    public static void main(String args[]) {  
        try {  
            String text = args[0];  
            System.out.println(text);  
        }  
        catch(Exception e) {  
            System.out.println("Hay nhap tham so khi chay!");  
        }  
    }  
}  
  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>java ArgExceptionDemo  
Hay nhap tham so khi chay!  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

Ví dụ không xử lý ngoại lệ

```
class NoException {  
    public static void main(String args[]) {  
        String text = args[0];  
        System.out.println(text);  
    }  
}
```



```
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>java NoException  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at NoException.main<NoException.java:3>  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

Ví dụ chia cho 0

```
public class ChiaCho0Demo {  
    public static void main(String args[]) {  
        try {  
            int num = calculate(9,0);  
            System.out.println(num);  
        }  
        catch(Exception e) {  
            System.err.println("Co loi xay ra: " + e.toString());  
        }  
    }  
    static int calculate(int no, int no1){  
        int num = no / no1;  
        return num;  
    }  
}  
D:\FIT-HUT\Lectures\OOP\OOP-Java\Demo>java ChiaCho0Demo  
Co loi xay ra: java.lang.ArithmeticException: / by zero  
Press any key to continue . . .
```

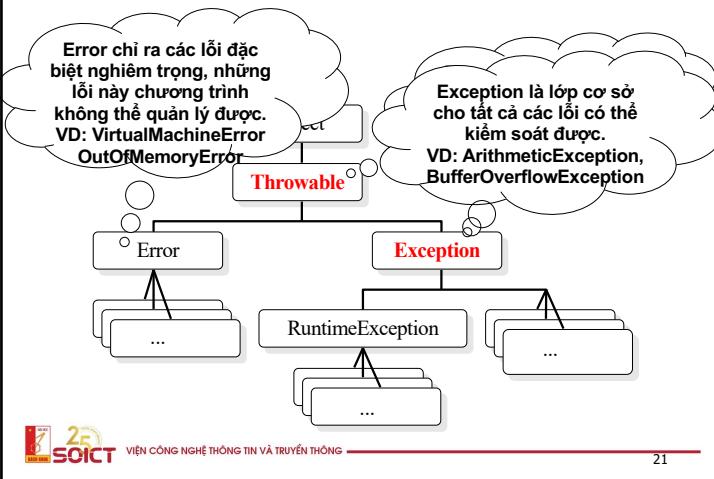


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

19

2.3.2. Cây phân cấp ngoại lệ trong Java



21

a. Lớp Throwable

- ❖ Một biến kiểu String để lưu thông tin chi tiết về ngoại lệ đã xảy ra
- ❖ Một số phương thức cơ bản
 - **new Throwable(String s)** : Tạo một ngoại lệ với thông tin về ngoại lệ là s
 - **String getMessage()** : Lấy thông tin về ngoại lệ
 - **void printStackTrace()** : In ra tất cả các thông tin liên quan đến ngoại lệ (tên, loại, vị trí...)
 - ...

```
public class StckExceptionDemo {  
    public static void main(String args[]) {  
        try {  
            int num = calculate(9,0);  
            System.out.println(num);  
        }  
        catch(Exception e) {  
            System.err.println("Co loi xay ra :" + e.getMessage());  
            e.printStackTrace();  
        }  
    }  
    static int calculate(int no, int no1) {  
        int num = no / no1;  
        return num;  
    }  
}  
Co loi xay ra :/ by zero  
java.lang.ArithmetricException: / by zero  
at StckExceptionDemo.calculate(StckExceptionDemo.java:14)  
at StckExceptionDemo.main(StckExceptionDemo.java:4)  
Press any key to continue . . .
```

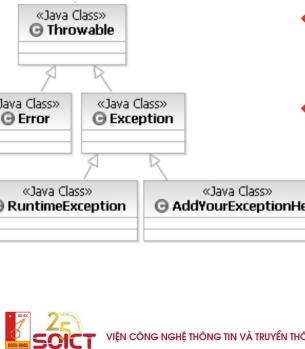
23

b. Lớp Error

- ❖ Gồm các ngoại lệ nghiêm trọng không thể kiểm tra (unchecked exception) vì có thể xảy ra ở nhiều phần của chương trình.
- ❖ Còn gọi là ngoại lệ không thể phục hồi (un-recoverable exception)
- ❖ Không cần kiểm tra trong mã nguồn Java của bạn
- ❖ Các lớp con:
 - VirtualMachineError: InternalError, OutOfMemoryError, StackOverflowError, UnknownError
 - ThreadDeath
 - LinkageError:
 - IncompatibleClassChangeError
 - AbstractMethodError, InstantiationException, NoSuchFieldError, NoSuchMethodError...
 - ...
 - ...

c. Lớp Exception

- ❖ Chứa các loại ngoại lệ nên/phải bắt và xử lý hoặc ủy nhiệm.
- ❖ Người dùng có thể tạo ra các ngoại lệ của riêng mình bằng cách kế thừa từ Exception
- ❖ RuntimeException có thể được “tung” ra trong quá trình JVM thực hiện
 - Không bắt buộc phải bắt ngoại lệ dù có thể xảy ra lỗi
 - Không nên viết ngoại lệ của riêng mình kế thừa từ lớp này



25

25

Ví dụ IOException

```
import java.io.InputStreamReader;
import java.io.IOException;
public class HelloWorld{
    public static void main(String[] args) {
        InputStreamReader isr = new
            InputStreamReader(System.in);
        try {
            System.out.print("Nhập vào 1 ký tự: ");
            char c = (char) isr.read();
            System.out.println("Ký tự vừa nhập: " + c);
        }catch(IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

Nhập vào 1 ký tự: b
Ký tự vừa nhập: b
Press any key to continue . . .



27

27

Một số lớp con của Exception

- ❖ ClassNotFoundException, SQLException
- ❖ java.io.IOException:
 - FileNotFoundException, EOFException...
- ❖ RuntimeException:
 - NullPointerException, BufferOverflowException
 - ClassCastException, ArithmeticException
 - IndexOutOfBoundsException:
 - ArrayIndexOutOfBoundsException,
 - StringIndexOutOfBoundsException...
 - IllegalArgumentException:
 - NumberFormatException, InvalidParameterException...
 - ...



26

26

2.3.3. Khối try – catch lồng nhau

- ❖ Những phần nhỏ trong khối mã sinh ra một lỗi, nhưng toàn bộ cả khối thì lại sinh ra một lỗi khác → Cần có các xử lý ngoại lệ lồng nhau.
- ❖ Khi các khối try lồng nhau, khối try bên trong sẽ được thực hiện trước.

```
try {
    // Đoán ma có thể gây ra IOException
    try {
        // Đoán ma có thể gây ra NumberFormatException
        catch (NumberFormatException e1) {
            // Xử lý lỗi sai định dạng số
        }
    } catch (IOException e2) {
        // Xử lý lỗi vào ra
    }
}
```



28

7

2.3.4. Nhiều khối catch

- Một đoạn mã có thể gây ra nhiều hơn một ngoại lệ → Sử dụng nhiều khối catch.

```
try {  
    // Đoạn ma co the gay ra nhieu hon mot ngoai le  
} catch (ExceptionType1 e1) {  
    // Xu ly ngoai le 1  
} catch (ExceptionType2 e2) {  
    // Xu ly ngoai le 2  
} ...
```

- ExceptionType1 phải là lớp con hoặc ngang hàng với ExceptionType2 (trong cây phân cấp kế thừa)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

29

- ExceptionType1 phải là lớp con hoặc ngang hàng với ExceptionType2 (trong cây phân cấp kế thừa)

```
class MultipleCatch1 {  
    public static void main(String args[]) {  
        try {  
            String num = args[0];  
            int numValue = Integer.parseInt(num);  
            System.out.println("Dien tich hv la: "  
                               + numValue * numValue);  
        } catch(ArrayIndexOutOfBoundsException e1) {  
            System.out.println("Hay nhap canh cua hv!");  
        } catch(NumberFormatException e2){  
            System.out.println("Hay nhap 1 so!");  
        }  
    }  
}
```

31

- ExceptionType1 phải là lớp con hoặc ngang hàng với ExceptionType2 (trong cây phân cấp kế thừa)

```
class MultipleCatch1 {  
    public static void main(String args[]) {  
        try {  
            String num = args[0];  
            int numValue = Integer.parseInt(num);  
            System.out.println("Dien tich hv la: "  
                               + numValue * numValue);  
        } catch(Exception e1) {  
            System.out.println("Hay nhap canh cua hv!");  
        } catch(NumberFormatException e2){  
            System.out.println("Not a number!");  
        }  
    }  
}
```

D:\exception java.lang.NumberFormatException has already been caught

30

30

```
class MultiCatch2 {  
    public static void main( String args[] ) {  
        try {  
            // format a number  
            // read a file  
            // something else...  
        }  
        catch(IOException e) {  
            System.out.println("I/O error "+e.getMessage());  
        }  
        catch(NumberFormatException e) {  
            System.out.println("Bad data "+e.getMessage());  
        }  
        catch(Throwable e) { // catch all  
            System.out.println("error: " + e.getMessage());  
        }  
    }  
}
```

32

31

```

...
public void openFile(){
    try {
        // constructor may throw FileNotFoundException
        FileReader reader = new FileReader("someFile");
        int i=0;
        while(i != -1) {
            //reader.read() may throw IOException
            i = reader.read();
            System.out.println((char) i );
        }
        reader.close();
        System.out.println("--- File End ---");
    } catch (FileNotFoundException e) {
        //do something clever with the exception
    } catch (IOException e) {
        //do something clever with the exception
    }
}
...

```

33

33

Cú pháp try ... catch ... finally

```

try {
    // Khoi lenh co the sinh ngoai le
} catch(ExceptionType e) {
    // Bat va xu ly ngoai le
} finally {
    /* Thuc hien cac cong viec can thiet
    du ngoai le co xay ra hay khong */
}

```

- ❖ Nếu đã có khối try thì bắt buộc phải có khối catch hoặc khối finally hoặc cả hai

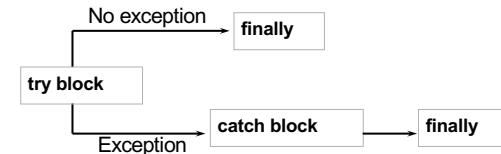


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

2.3.5. Khối finally

- ❖ Đảm bảo thực hiện tất cả các công việc cần thiết khi có ngoại lệ xảy ra
 - Đóng file, đóng socket, connection
 - Giải phóng tài nguyên (nếu cần)...
- ❖ Chắc chắn sẽ thực hiện dù ngoại lệ có xảy ra hay không.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

34

```

class StrExceptionDemo {
    static String str;
    public static void main(String s[]) {
        try {
            System.out.println("Truoc ngoai le");
            staticLengthmethod();
            System.out.println("Sau ngoai le");
        } catch(NullPointerException ne) {
            System.out.println("Da xay ra loi");
        } finally {
            System.out.println("Trong finally");
        }
    }
    static void staticLengthmethod() {
        System.out.println(str.length());
    }
}

```

Truoc ngoai le
 Da xay ra loi
 Khoi finally

36

35

```

public void openFile(){
    try {
        // constructor may throw FileNotFoundException
        FileReader reader = new FileReader("someFile");
        int i=0;
        while(i != -1) {
            //reader.read() may throw IOException
            i = reader.read();
            System.out.println((char) i );
        }
    } catch (FileNotFoundException e) {
        //do something clever with the exception
    } catch (IOException e) {
        //do something clever with the exception
    } finally {
        reader.close();
        System.out.println("---- File End ----");
    }
}

```

37

37

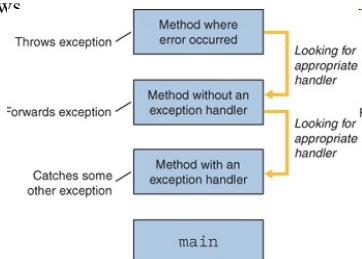
Nội dung

1. Ngoại lệ
2. Bắt và xử lý ngoại lệ
- 3. Ủy nhiệm ngoại lệ**
4. Tạo ngoại lệ tự định nghĩa

38

Hai cách làm việc với ngoại lệ

- ❖ Xử lý ngay
 - Sử dụng khối try ... catch (finally nếu cần).
- ❖ Ủy nhiệm cho vị trí gọi nó:
 - Nếu không muốn xử lý ngay
 - Sử dụng throw và throws



39

3.1. Ủy nhiệm ngoại lệ

- ❖ Phương thức có thể ủy nhiệm ngoại lệ cho vị trí gọi nó bằng cách:
 - Sử dụng throws ExceptionType ở phần khai báo phương thức để báo hiệu cho vị trí gọi nó biết là nó có thể phát sinh ngoại lệ ExceptionType
 - Sử dụng throw để tung ra ngoại lệ kiểu ExceptionType trong thân phương thức khi cần

Ví dụ

```

public void myMethod(int param) throws
    Exception{
    if (param < 10) {
        throw new Exception("Too low!");
    }
    //Blah, Blah, Blah...
}

```

40

3.1. Ủy nhiệm ngoại lệ (2)

- Nếu phương thức có chứa câu lệnh tung ngoại lệ (throw) thì phần khai báo phương thức phải khai báo là có tung ngoại lệ đó hoặc lớp cha của ngoại lệ đó

```
public void myMethod(int param) {  
    if (param < 10) {  
        throw new Exception("Too low!");  
    }  
    //Blah, Blah, Blah...  
}  
→ unreported exception java.lang.Exception; must be  
caught or declared to be thrown
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

3.1. Ủy nhiệm ngoại lệ (3)

- Phương thức không cần phải khai báo sẽ tung ra RuntimeException vì ngoại lệ này mặc định được ủy nhiệm cho JVM

- Ví dụ

```
class Test {  
    public void myMethod(int param) {  
        if (param < 10) {  
            throw new RuntimeException("Too  
low!");  
        }  
        //Blah, Blah, Blah...  
    }  
}
```

→ Không lỗi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

3.1. Ủy nhiệm ngoại lệ (3)

- Tại vị trí gọi phương thức có ủy nhiệm ngoại lệ (trừ RuntimeException):
 - Hoặc là phương thức chứa vị trí đó phải ủy nhiệm tiếp cho vị trí gọi mình
 - Hoặc là tại vị trí gọi phải bắt ngoại lệ ủy nhiệm (hoặc lớp cha) và xử lý ngay bằng try...catch (finally nếu cần)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

```
public class DelegateExceptionDemo {  
    public static void main(String args[]) {  
        int num = calculate(9, 3);  
        System.out.println("Lan 1: " + num);  
        num = calculate(9, 0);  
        System.out.println("Lan 2: " + num);  
    }  
    static int calculate(int no, int no1)  
        throws ArithmeticException {  
        if (no1 == 0)  
            throw new  
                ArithmeticException("Khong the chia cho 0!");  
        int num = no / no1;  
        return num;  
    }  
}
```

Lan 1: 3
Exception in thread "main" java.lang.ArithmeticException: Khong the chia cho 0!
at DelegateExceptionDemo.calculate(DelegateExceptionDemo.java:11)
at DelegateExceptionDemo.main(DelegateExceptionDemo.java:5)
Press any key to continue . . .



44

```

public class DelegateExceptionDemo {
    public static void main(String args[]){
        int num = calculate(9,3);
        System.out.println("Lan 1: " + num);
        num = calculate(9,0);
        System.out.println("Lan 2: " + num);
    }
    static int calculate(int no, int nol)
            throws Exception {
        if (nol == 0)
            throw new
                    ArithmeticException("Khong the chia cho 0!");
        int num = no / nol;
        return num;
    }
}
G:\Java Example\DelegateExceptionDemo.java:3: unreported exception java.lang.Exception;
must be caught or declared to be thrown
    int num = calculate(9,3);
               ^
G:\Java Example\DelegateExceptionDemo.java:5: unreported exception java.lang.Exception;
must be caught or declared to be thrown
    num = calculate(9,0);
               45

```

Compile

45

```

public class DelegateExceptionDemo {
    public static void main(String args[]){
        try {
            int num = calculate(9,3);
            System.out.println("Lan 1: " + num);
            num = calculate(9,0);
            System.out.println("Lan 2: " + num);
        } catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
    static int calculate(int no, int nol)
            throws ArithmeticException {
        if (nol == 0)
            throw new
                    ArithmeticException("Khong the chia cho 0!");
        int num = no / nol;
        return num;
    }
}
Lan 1: 3
Khong the chia cho 0!
Press any key to continue . . . - 46

```

46

3.1. Ủy nhiệm ngoại lệ (4)

- Một phương thức có thể ủy nhiệm nhiều hơn 1 ngoại lệ

```

public void myMethod(int tuoi, String ten)
        throws ArithmeticException, NullPointerException{
    if (tuoi < 18) {
        throw new ArithmeticException("Chua du tuoi!");
    }
    if (ten == null) {
        throw new NullPointerException("Thieu ten!");
    }
    //Blah, Blah, Blah...
}

```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

47

3.2. Lan truyền ngoại lệ

- Tình huống:

- Giả sử trong main() gọi phương thức A(), trong A() gọi B(), trong B() gọi C(). Khi đó một ngăn xếp các phương thức được tạo ra.
- Giả sử trong C() xảy ra ngoại lệ.

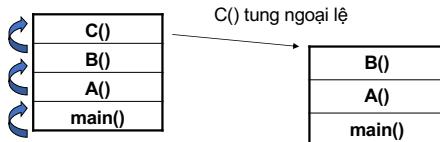


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

12

3.2. Lan truyền ngoại lệ (2)



- ❖ Nếu C() gặp lỗi và tung ra ngoại lệ nhưng trong C() lại không xử lý ngoại lệ này, thì chỉ còn một nơi có thể xử lý chính là nơi mà C() được gọi, đó là trong phương thức B().
- ❖ Nếu trong B() cũng không xử lý thì phải xử lý ngoại lệ này trong A()... Quá trình này gọi là lan truyền ngoại lệ
- ❖ Nếu đến main() cũng không xử lý ngoại lệ được tung từ C() thì chương trình sẽ phải dừng lại.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

3.3. Kế thừa và ủy nhiệm ngoại lệ (2)

```
class Disk {  
    void readFile() throws EOFException {}  
}  
  
class FloppyDisk extends Disk {  
    void readFile() throws IOException {} // ERROR!  
}  
  
class Disk {  
    void readFile() throws IOException {}  
}  
  
class FloppyDisk extends Disk {  
    void readFile() throws EOFException {} //OK  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

3.3. Kế thừa và ủy nhiệm ngoại lệ

- ❖ Khi override một phương thức của lớp cha, phương thức ở lớp con không được phép tung ra các ngoại lệ mới
- ❖ → Phương thức ghi đè trong lớp con chỉ được phép tung ra các ngoại lệ giống hoặc là lớp con hoặc là tập con của các ngoại lệ được tung ra ở lớp cha.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

3.4. Ưu điểm của ủy nhiệm ngoại lệ

- ❖ Dễ sử dụng
 - Làm chương trình dễ đọc và an toàn hơn
 - Dễ dàng chuyển điều khiển đến nơi có khả năng xử lý ngoại lệ
 - Có thể ném nhiều loại ngoại lệ
- ❖ Tách xử lý ngoại lệ khỏi đoạn mã thông thường
- ❖ Không bỏ sót ngoại lệ (ném tự động)
- ❖ Gom nhóm và phân loại các ngoại lệ
- ❖ KL: **Làm chương trình dễ đọc và an toàn hơn**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

51

13

Nội dung

1. Ngoại lệ
2. Bắt và xử lý ngoại lệ
3. Ủy nhiệm ngoại lệ
- 4. Tạo ngoại lệ tự định nghĩa**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

Sử dụng ngoại lệ người dùng định nghĩa

Khai báo khả năng tung ngoại lệ

```
public class FileExample
{
    public void copyfile(String fName1, String fName2)
    throws MyException
    {
        if (fName1.equals(fName2))
            throw new MyException("File trùng tên");
        // Copy file
        System.out.println("Copy completed");
    }
}
```

Tung ngoại lệ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

55

4. Tạo ngoại lệ tự định nghĩa

- ❖ Các ngoại lệ do hệ thống xây dựng không đủ để kiểm soát tất cả các lỗi → Cần phải có các lớp ngoại lệ do người dùng định nghĩa.

- Kế thừa từ một lớp **Exception** hoặc lớp con của nó
- Có tất cả các phương thức của lớp **Throwable**

```
public class MyException extends Exception {
    public MyException(String msg) {
        super(msg);
    }
    public MyException(String msg, Throwable cause) {
        super(msg, cause);
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

54

Sử dụng ngoại lệ người dùng định nghĩa

Bắt và xử lý ngoại lệ

```
public class Test {
    public static void main(String[] args) {
        FileExample obj = new FileExample();
        try {
            String a = args[0];
            String b = args[1];
            obj.copyfile(a,b);
        } catch (MyException e1) {
            System.out.println(e1.getMessage());
        }
        catch (Exception e2) {
            System.out.println(e2.toString());
        }
    }
}
```



VIỆN CÔNG NGHỆ THÔNG

```
C:\>java Test a1.txt a1.txt
File trùng tên
C:\>java Test
java.lang.ArrayIndexOutOfBoundsException: 0
```

6

56

14

Tổng kết

- ❖ Bất cứ khi nào có một lỗi xảy ra khi thực hiện chương trình thì một ngoại lệ đã xuất hiện.
- ❖ Mọi ngoại lệ đều phải được xử lý nếu không muốn chương trình kết thúc một cách bất thường.
- ❖ Xử lý ngoại lệ cho phép kết hợp và xử lý lỗi tại một nơi.
- ❖ Java sử dụng khối try/catch để quản lý ngoại lệ.

Tổng kết (2)

- ❖ Các câu lệnh trong khối try tung ra ngoại lệ, và việc xử lý các ngoại lệ đó diễn ra trong khối catch.
- ❖ Nhiều khối catch có thể được sử dụng để xử lý tách biệt các loại ngoại lệ khác nhau.
- ❖ Từ khóa throws được sử dụng để liệt kê mỗi danh sách các ngoại lệ mà một phương thức có thể tung ra.
- ❖ Từ khóa throw được sử dụng để tung ra một ngoại lệ.
- ❖ Khối finally để thực hiện các công việc cần thiết dù có ngoại lệ xảy ra hay không.

Bài tập

Bài tập 1

- ❖ Hệ thống liên tục nhận các giá trị đầu vào là xâu đại diện cho một số nguyên, yêu cầu mỗi lần nhận được một số thì tính trung bình cộng của các giá trị đã nhận.
- ❖ Xây dựng phương thức:
 - public double getAverage(string x) với thuộc tính average, N có sẵn trong lớp
- ❖ Hãy cài đặt phương thức trong đó có xử lý các ngoại lệ xảy ra:
 - Xâu đầu vào là xâu rỗng
 - Xâu đầu vào không phải là số
 - Xâu đầu vào không phải là số nguyên

Bài tập 2

- ❖ Hệ thống cần đọc file để lấy ra một dãy các con số (mỗi dòng một số nguyên)
 - Dòng 1 của file là số lượng các số có trong file
 - Mỗi dòng là một số nguyên
- ❖ Xây dựng phương thức đọc dãy số:
 - public void readListIntegers(String fileName)
- ❖ Hãy xử lý các ngoại lệ:
 - Xâu tên file là xâu rỗng
 - Không tìm thấy file
 - Không mở được file
 - Các xâu trong từng dòng của file không phải là đại diện cho con số



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

61

Bài tập 3

- ❖ Hệ thống cần đọc file đầu vào để lấy ra một dãy các con số (mỗi dòng một số nguyên), sau đó tách ra thành 4 phần bằng nhau và ghi ra các file khác nhau.
 - Dòng 1 của file đầu vào là số lượng các số có trong file. Mỗi dòng sau đó là một số nguyên
 - Xây dựng phương thức phân tách dãy số:
 - public void splitListIntegers(String fileName)
- ❖ Hãy xử lý các ngoại lệ:
 - Xâu tên file là xâu rỗng
 - Không tìm thấy file
 - Không mở được file
 - Các xâu trong từng dòng của file không phải là đại diện cho con số
 - Không ghi được file mới



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

62

16



Bài 11: Lập trình giao diện với JavaFX

1

Bài giảng Elearning

- ❖ JavaFx Tutorial For Beginners

https://www.youtube.com/watch?v=9YrmON6nIew&list=PLS1QulWo1RlaUGP446_pWLgTZPiFizEMg

- ❖ Khóa học lập trình JavaFX

<https://www.youtube.com/watch?v=zAg7Lmv46PE&list=PL33lvabfss1yRgFCgFXjtYaGAuDJjH-j>



2

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder



3

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder



4

1. Giới thiệu

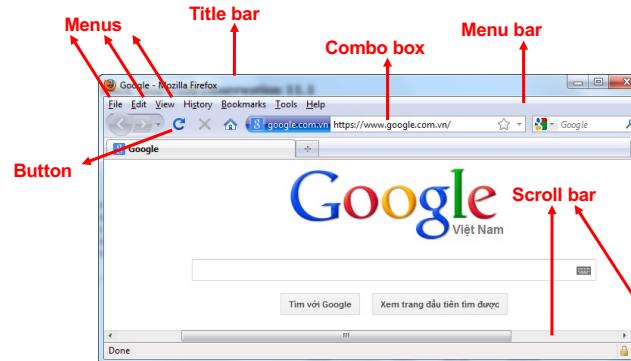
- ❖ Giao diện đồ họa người dùng: Graphical user interface - GUI (pronounced "GOO-ee"):
 - Là một loại giao diện người dùng
 - Cho phép người dùng tương tác với các thiết bị điện tử, sử dụng hình ảnh thay vì nhập vào các lệnh
- ❖ Tại sao sử dụng thuật ngữ GUI?
 - Giao diện tương tác người dùng đầu tiên là giao diện dòng lệnh



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

1. Giới thiệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

Java APIs cho lập trình đồ họa

- ❖ AWT (Abstract Windowing Toolkit)
 - Được giới thiệu trong JDK 1.0
 - Không nên dùng, dùng Swing thay thế
- ❖ Swing:
 - Mở rộng AWT
 - Tích hợp vào Java từ JDK 1.2
- ❖ JavaFX:
 - Thư viện Java, phát triển ứng dụng đa nền tảng (Desktop, mobile, TV, tablet)
- ❖ Các thư viện khác:
 - Eclipse's Standard Widget Toolkit (SWT)
 - Google Web Toolkit (GWT)
 - 3D Graphics API: Java OpenGL (JOGL), Java3D.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

JavaFX – Tính năng (Features)

- ❖ Viết bằng Java, dùng được trong các ngôn ngữ thực thi trên máy ảo Java (Java, Groovy và JRuby)
- ❖ Hỗ trợ FXML (tương tự HTML), giúp dễ dàng định nghĩa giao diện người dùng
- ❖ Scene Builder: JavaFX cung cấp ứng dụng Scene Builder trên các nền tảng khác nhau, cho phép LTV kéo thả khi thiết kế giao diện
- ❖ Tương thích với Swing: trong ứng dụng JavaFX có thể nhúng các thành phần Swing
- ❖ Built-in UI controls: JavaFX cung cấp các control đa dạng để phát triển ứng dụng
- ❖ CSS like Styling: thiết kế giao diện với các tính năng giống như trong CSS
- ❖ ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

Lịch sử JavaFX

- ❖ JavaFX được phát triển bởi Chris Oliver khi ông làm trong tập đoàn See Beyond Technology Corporation (Được Sun Microsystems mua lại vào 2005)
- ❖ 2007: Được giới thiệu chính thức ở hội nghị Java One
- ❖ 2008: Được tích hợp vào NetBean. JavaFX 1.0 được ban hành
- ❖ 2014: JavaFX được tích hợp vào Java SDK 8
- ❖ 2018: JavaFX được tách ra khỏi Java SDK 11

2. Cài đặt JavaFX

- ❖ Trang chủ JavaFX: <https://openjfx.io/>
- ❖ Trang download thư viện JavaFX: <https://gluonhq.com/products/javafx/>
- ❖ Download, giải nén, copy các file trong thư mục lib, add vào build path của project
- ❖ Lưu ý khi chạy chương trình trên IDE Eclipse
 - Vào runtime configuration, cấu hình VM arguments:
 - -module-path \${project_classpath:REPLACE_ME_WITH_YOUR_PROJECT_NAME} --add-modules javafx.controls,javafx.fxml
 - Bỏ chọn: "Use the -XstartOnFirstThread argument when launching with SWT"

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder

JavaFX Hello World

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



Tiện ích JavaFX trên Eclipse

e(fx)clipse

- <https://www.eclipse.org/efxclipse/releases.html>
- Là một Eclipse plugin
- Công cụ hỗ trợ lập trình JavaFX trên Eclipse

JavaFX Scene Builder

- <https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html>
- Công cụ độc lập, đa nền tảng, thiết kế trực quan giao diện cho ứng dụng JavaFX.
- Cho phép kéo thả các thành phần giao diện người dùng, thay đổi thuộc tính, áp dụng style
- Đầu ra: file FXML dùng trong ứng dụng JavaFX

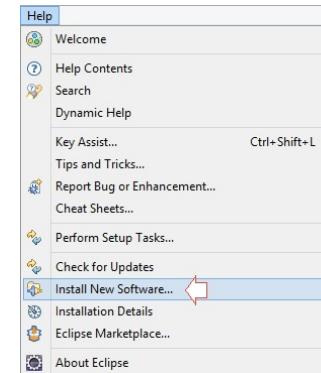


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

Cài đặt e(fx)clipse

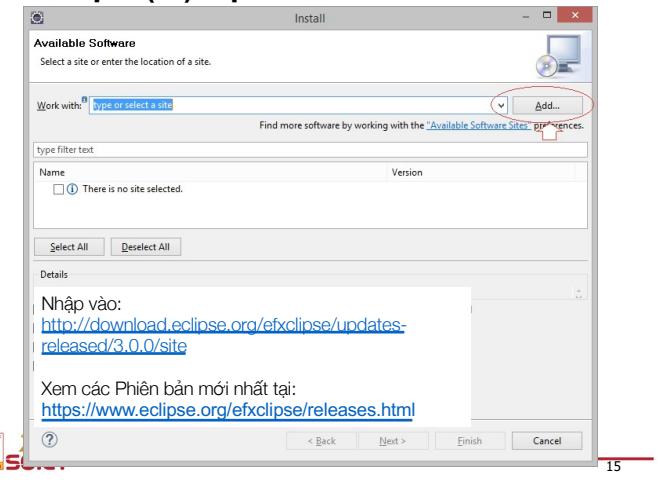


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

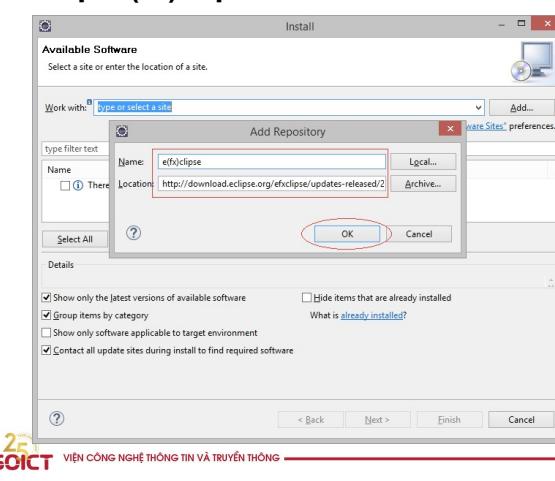
Cài đặt e(fx)clipse



15

15

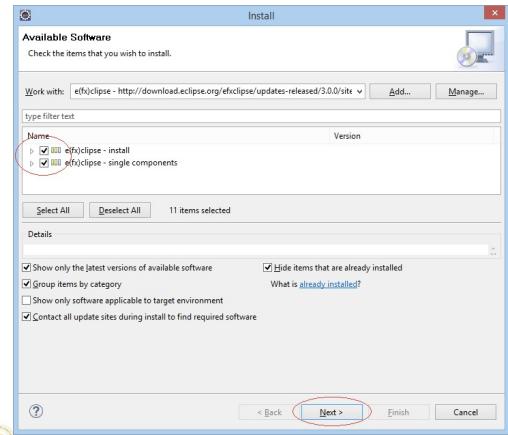
Cài đặt e(fx)clipse



16

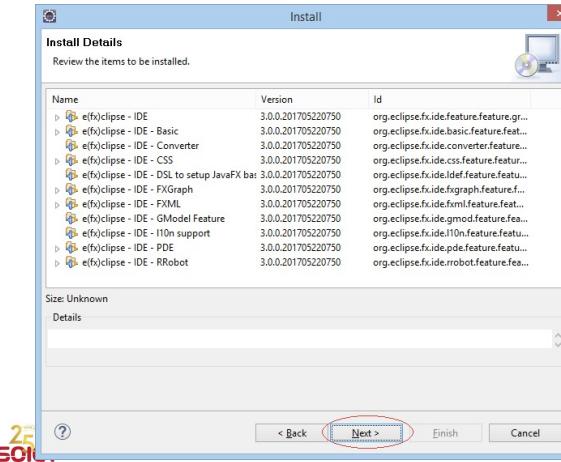
16

Cài đặt e(fx)clipse



17

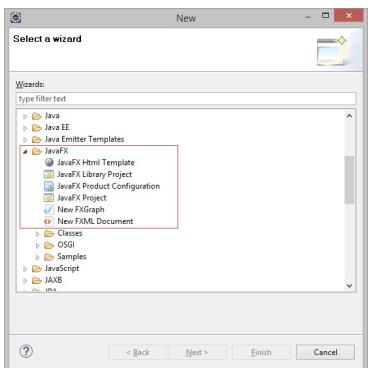
Cài đặt e(fx)clipse



18

Cài đặt e(fx)clipse

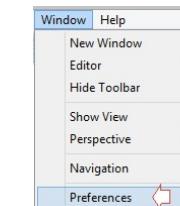
- Sau khi cài đặt và khởi động lại Eclipse, vào menu File/New/Others ... sẽ thấy các Wizard cho phép lập trình JavaFX



19

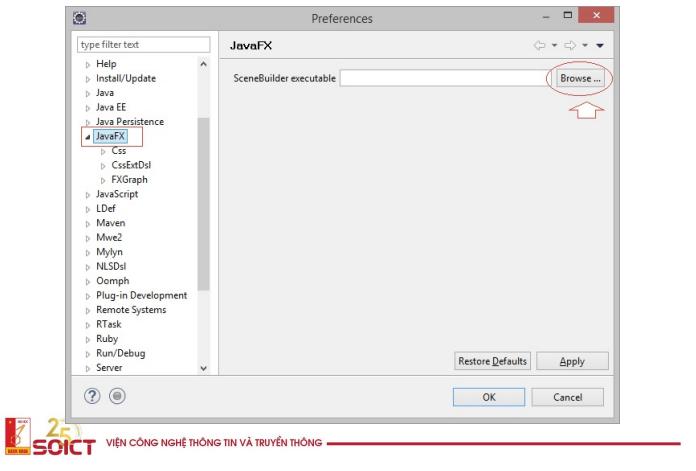
Tích hợp JavaFX Scene Builder vào Eclipse

- Download, cài đặt JavaFX Scene Builder
- Trên eclipse, vào Window/Preferences



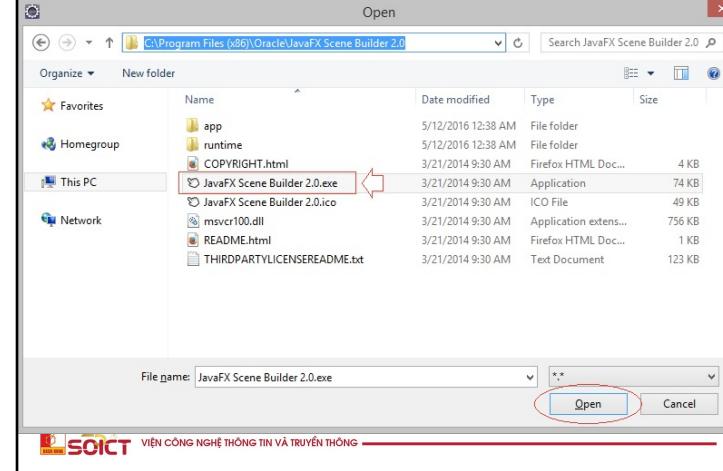
20

Tích hợp JavaFX Scene Builder vào Eclipse



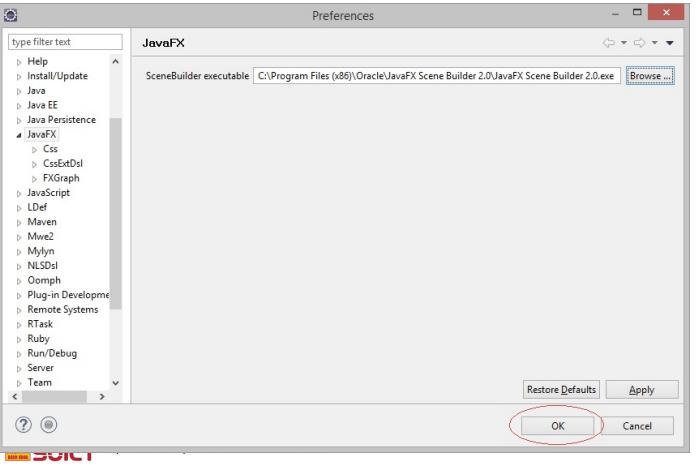
21

Tích hợp JavaFX Scene Builder vào Eclipse



22

Tích hợp JavaFX Scene Builder vào Eclipse



23

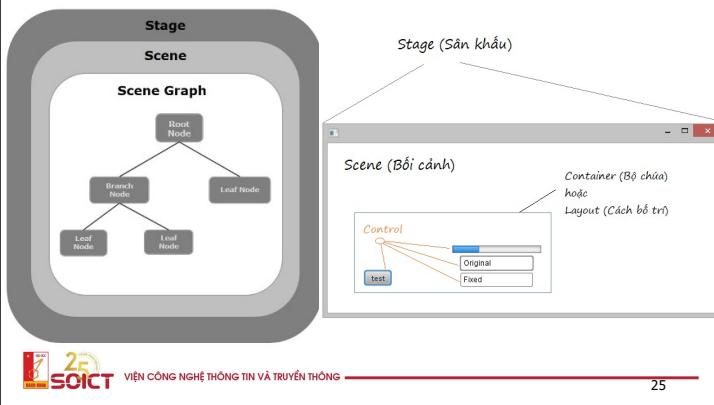
Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. **Các thành phần giao diện JavaFX**
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder

24

3. Các thành phần giao diện JavaFX

- ❖ Cấu trúc ứng dụng JavaFX gồm 3 thành phần chính: Stage, Scene và Nodes



25

25

Stage

- ❖ Đối tượng Stage (Window) chứa tất cả các đối tượng khác trong ứng dụng JavaFX
- ❖ Là đối tượng của lớp javafx.stage.Stage
- ❖ Đối tượng Stage sẽ truyền làm tham số cho phương thức start() của lớp Application (Xem lại ví dụ HelloWorld JavaFX)
- ❖ Có 2 tham số width và height
- ❖ Được chia làm 2 phần: Content Area và Decorations (Title bar và Borders)
- ❖ Để hiển thị Stage, gọi phương thức show()
- ❖ Có 5 style cho Stage: Decorated, Undecorated, Transparent, Unified, Utility



26

26

Stage – thiết lập style

```
stage.initStyle(StageStyle.DECORATED);  
//stage.initStyle(StageStyle.UNDECORATED);  
//stage.initStyle(StageStyle.TRANSPARENT);  
//stage.initStyle(StageStyle.UNIFIED);  
//stage.initStyle(StageStyle.UTILITY);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

27

JavaFX Hello World

```
import javafx.application.Application;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.StackPane;  
import javafx.stage.Stage;  
  
public class HelloWorld extends Application {  
    @Override  
    public void start(Stage primaryStage) {  
        Button btn = new Button();  
        btn.setText("Say 'Hello World'");  
        btn.setOnAction(new EventHandler<ActionEvent>() {  
            @Override  
            public void handle(ActionEvent event) {  
                System.out.println("Hello World!");  
            }  
        });  
        StackPane root = new StackPane();  
        root.getChildren().add(btn);  
        Scene scene = new Scene(root, 300, 250);  
        primaryStage.setTitle("Hello World!");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```



Scene

- ❖ Scene chứa tất cả các nội dung trình bày của một **scene graph**
- ❖ Là đối tượng của lớp javafx.scene.Scene
- ❖ Một Scene được thêm vào duy nhất một Stage
- ❖ Một số phương thức khởi tạo:
 - Scene(Parent root)
 - Scene(Parent root, double width, double height)
 - ...

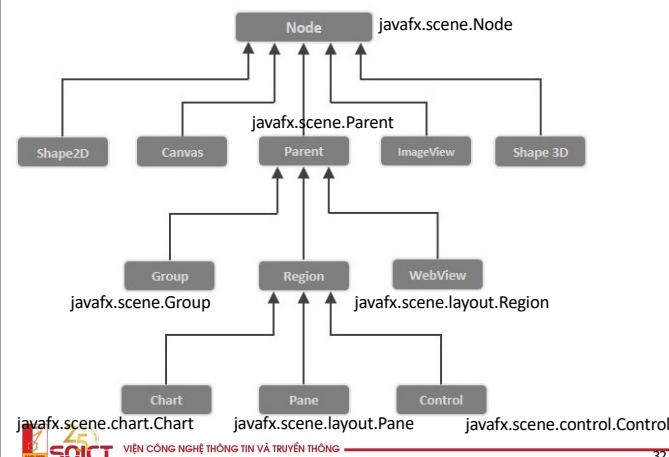
Scene Graph và Nodes

- ❖ **Scene graph:** là cấu trúc dữ liệu phân cấp dạng tree biểu diễn nội dung một **Scene**, bao gồm tất cả các controls, layout
- ❖ **Node:** là một đối tượng đồ họa của một Scene graph, bao gồm
 - Đối tượng hình học (2D và 3D) như: Circle, Rectangle, Polygon, ...
 - Đối tượng điều khiển UI như: Button, Checkbox, TextArea, ...
 - Phần tử đa phương tiện Media như: Audio, Video, Image
- ❖ Lớp cơ sở cho tất cả các loại Node: javafx.scene.Node

Scene Graph và Nodes

- ❖ Có 2 loại Node:
 - Branch Node/Parent Node: là các node có các node con, lớp cơ sở là lớp javafx.scene.Parent (lớp trừu tượng). Có 3 loại:
 - **Group:** là một node tổng hợp, chứa một list các node con. Khi render node Group, tất cả các node con sẽ lần lượt được render. Các chuyển đổi hiệu ứng áp dụng cho một Group được áp dụng cho tất cả node con
 - **Region:** là lớp cơ sở cho các UI Controls, bao gồm **Chart** (AreaChart, BarChart, BubbleChart, ...), **Pane** (AnchorPane, BorderPane, DialogPane, FlowPane, HBox, VBox ...), **Control** (Accordion, ButtonBar, ChoiceBox, ComboBoxBase, HTMLEditor, ...)
 - **WebView:** tương tự như Browser
 - Leaf Node: là node không có node con. Ví dụ: Rectangle, Ellipse, Box, ImageView, MediaView
- ❖ Lưu ý: Root node là một branch/parent node, nhưng root node không có node cha.

Cây phân cấp kế thừa Node



Cách tạo ứng dụng JavaFX

- ❖ Viết lớp kế thừa lớp javafx.application.Application, thực thi phương thức trừu tượng start
- ❖ Trong phương thức main, gọi phương thức static launch(). Phương thức launch đã tự động gọi phương thức start()

```
public class JavaFxSample extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        /*  
         * Code for JavaFX application.  
         * (Stage, scene, scene graph)  
         */  
    }  
    public static void main(String args[]) {  
        launch(args);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

Cài đặt phương thức start

- ❖ 3 bước:
 - Tao một Scene graph với các Node
 - Tao một Scene với kích thước mong muốn và thêm vào root node của scene graph
 - Tao một Stage, thêm Scene vào Stage, và hiển thị nội dung của Stage



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

Vòng đời ứng dụng JavaFX

- ❖ Có 3 phương thức trong vòng đời ứng dụng JavaFX: start(), stop(), init()
- ❖ Cài đặt mặc định là phương thức rỗng, có thể override khi muốn làm gì đó
- ❖ Thứ tự hành động
 - Tạo thể hiện của lớp application
 - Gọi phương thức init (không tạo stage hoặc scene trong phương thức này)
 - Gọi phương thức start
 - Khi ứng dụng kết thúc, gọi phương thức stop
- ❖ Khi cửa sổ (window) cuối cùng của ứng dụng JavaFX được đóng, ứng dụng tự động kết thúc. Có thể gọi tường minh với phương thức Platform.exit() hoặc System.exit(int)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

Tạo scene graph

- ❖ Cần tạo node gốc, có thể là Group, Region hoặc WebView
 - VD: Group root = new Group();
- ❖ Thêm các node vào root node theo 2 cách
 - Cách 1:

```
//Retrieving the observable list object  
ObservableList list = root.getChildren();  
  
//Setting a node object as a node  
list.add(NodeObject);
```
 - Cách 2:

```
Group root = new Group(NodeObject);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

33

34

35

Tạo Scene

- ❖ Khởi tạo đối tượng Scene, bắt buộc phải truyền tham số là root object

```
Scene scene = new Scene(root);
```

- ❖ Có thể vừa khởi tạo vừa thiết lập kích thước của Scene

```
Scene scene = new Scene(root, 600, 300);
```

Ví dụ: tạo ứng dụng với cửa sổ JavaFX rỗng

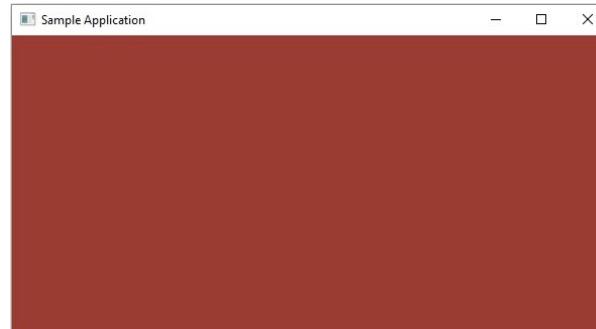
```
public class JavafxSample extends Application {  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        //creating a Group object  
        Group group = new Group();  
  
        //Creating a Scene  
        Scene scene = new Scene(group, 600, 300);  
  
        //setting color to the scene  
        scene.setFill(Color.BROWN);  
  
        //Setting the title to Stage.  
        primaryStage.setTitle("Sample Application");  
  
        //Adding the scene to Stage  
        primaryStage.setScene(scene);  
  
        //Displaying the contents of the stage  
        primaryStage.show();  
    }  
    public static void main(String args[]) {  
        launch(args);  
    }  
}
```

Tạo Stage

- ❖ Đối tượng Stage được truyền làm tham số cho phương thức start() của lớp Application → không cần khởi tạo
- ❖ Thao tác cơ bản

```
//Setting the title to Stage.  
primaryStage.setTitle("Sample application");  
  
//Setting the scene to Stage  
primaryStage.setScene(scene);  
  
//Displaying the stage  
primaryStage.show();
```

Ví dụ: tạo ứng dụng với cửa sổ JavaFX rỗng



VD: vẽ đường thẳng

```
public class DrawingLine extends Application{  
    @Override  
    public void start(Stage stage) {  
        //Creating a line object  
        Line line = new Line();  
  
        //Setting the properties to a line  
        line.setStartX(100.0);  
        line.setStartY(150.0);  
        line.setEndX(500.0);  
        line.setEndY(150.0);  
  
        //Creating a Group  
        Group root = new Group(line);  
  
        //Creating a Scene  
        Scene scene = new Scene(root, 600, 300);  
  
        //Setting title to the scene  
        stage.setTitle("Sample application");  
  
        //Adding the scene to the stage  
        stage.setScene(scene);  
  
        //Displaying the contents of a scene  
        stage.show();  
    }  
    public static void main(String args[]){  
        launch(args);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

VD: vẽ đường thẳng



42

VD: hiển thị dòng chữ

```
public class DisplayingText extends Application {  
    @Override  
    public void start(Stage stage) {  
        //Creating a Text object  
        Text text = new Text();  
  
        //Setting font to the text  
        text.setFont(new Font(45));  
  
        //setting the position of the text  
        text.setX(50);  
        text.setY(150);  
  
        //Setting the text to be added.  
        text.setText("Welcome to Tutorialspoint");  
  
        //Creating a Group object  
        Group root = new Group();  
  
        //Retrieving the observable list object  
        ObservableList list = root.getChildren();  
  
        //Setting the text object as a node to the group object  
        list.add(text);  
  
        //Creating a scene object  
        Scene scene = new Scene(root, 600, 300);  
  
        //Setting title to the Stage  
        stage.setTitle("Sample Application");  
  
        //Adding scene to the stage  
        stage.setScene(scene);  
  
        //Displaying the contents of the stage  
        stage.show();  
    }  
    public static void main(String args[]){  
        launch(args);  
    }  
}
```

43

VD: hiển thị dòng chữ



44

Ví dụ: hiển thị 2 dòng text

```
public class DecorationsExample extends Application {  
    @Override  
    public void start(Stage stage) {  
        //Creating a Text Example object  
        Text text1 = new Text("Hi how are you");  
        //Setting font to the text  
        text1.setFont(  
            Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20)  
        );  
        //setting the position of the text  
        text1.setX(50);  
        text1.setY(75);  
        //Striking through the text  
        text1.setStrikeThrough(true);  
        //Creating a Text Example object  
        Text text2 = new Text("Welcome to Tutorialspoint");  
        //Setting font to the text  
        text2.setFont(  
            Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20)  
        );  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

Ví dụ: hiển thị 2 dòng text

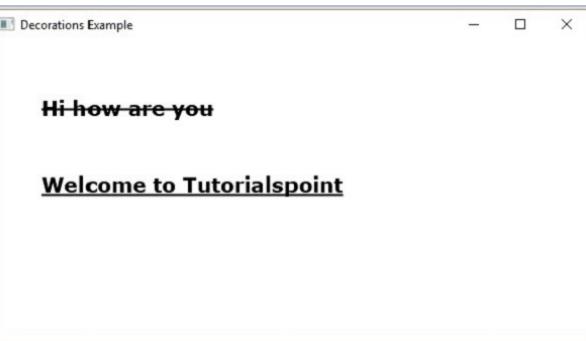
```
//setting the position of the text  
text2.setX(50);  
text2.setY(150);  
//underlining the text  
text2.setUnderline(true);  
//Creating a Group object  
Group root = new Group(text1, text2);  
//Creating a scene object  
Scene scene = new Scene(root, 600, 300);  
//Setting title to the Stage  
stage.setTitle("Decorations Example");  
//Adding scene to the stage  
stage.setScene(scene);  
//Displaying the contents of the stage  
stage.show();  
}  
public static void main(String args[]){  
    launch(args);  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

Ví dụ: hiển thị 2 dòng text



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

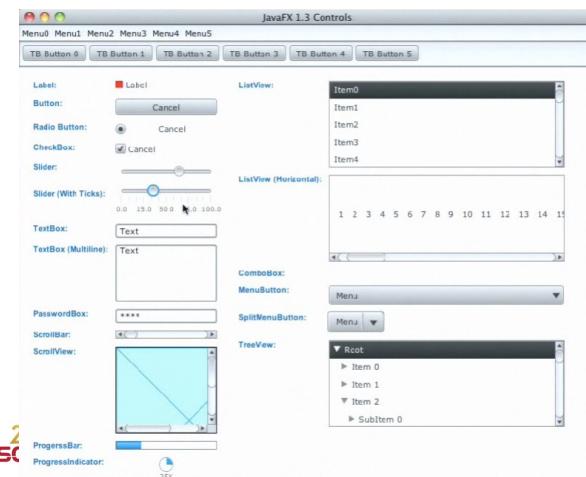
4. Java FX - UI Controls

- Một giao diện người dùng gồm 3 thành phần chính
- ❖ **UI elements** – Là các phần tử người dùng thấy sau cùng và trực tiếp tương tác với (Button, Label, Checkbox, ...)
 - ❖ **Layouts** – Định nghĩa cách thức sắp xếp các UI elements trên màn hình
 - ❖ **Behavior** – Các sự kiện xảy ra khi người dùng tương tác với các UI elements (Event Handling)

4. Java FX - UI Controls



4. Java FX - UI Controls



Ví dụ (1/5)

- ❖ Viết ứng dụng với giao diện như sau



Ví dụ (2/5)

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;

import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

Ví dụ (3/5)

```
public class LoginPage extends Application {
    @Override
    public void start(Stage stage) {
        //creating label email
        Text text1 = new Text("Email");

        //creating label password
        Text text2 = new Text("Password");

        //Creating Text Filed for email
        TextField textField1 = new TextField();

        //Creating Text Filed for password
        PasswordField textField2 = new PasswordField();

        //Creating Buttons
        Button button1 = new Button("Submit");
        Button button2 = new Button("Clear");
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

Ví dụ (4/5)

```
//Creating a Grid Pane
GridPane gridPane = new GridPane();

//Setting size for the pane
gridPane.setMinSize(400, 200);

//Setting the padding
gridPane.setPadding(new Insets(10, 10, 10, 10));

//Setting the vertical and horizontal gaps between the columns
gridPane.setVgap(5);
gridPane.setHgap(5);

//Setting the Grid alignment
gridPane.setAlignment(Pos.CENTER);

//Arranging all the nodes in the grid
gridPane.add(text1, 0, 0);
gridPane.add(textField1, 1, 0);
gridPane.add(text2, 0, 1);
gridPane.add(textField2, 1, 1);
gridPane.add(button1, 0, 2);
gridPane.add(button2, 1, 2);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

Ví dụ (5/5)

```
//Styling nodes
button1.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");
button2.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");

text1.setStyle("-fx-font: normal bold 20px 'serif' ");
text2.setStyle("-fx-font: normal bold 20px 'serif' ");
gridPane.setStyle("-fx-background-color: BEIGE;");

//Creating a scene object
Scene scene = new Scene(gridPane);

//Setting title to the Stage
stage.setTitle("CSS Example");

//Adding scene to the stage
stage.setScene(scene);

//Displaying the contents of the stage
stage.show();
}

public static void main(String args[]){
    launch(args);
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

55

14

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
- 5. JavaFX - Layout Panes**
6. Mô hình xử lý sự kiện
7. Kéo thả giao diện với SceneBuilder



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

57

Các bước tạo Layout

- ❖ Tạo các nodes
- ❖ Khởi tạo đối tượng của lớp layout mong muốn
- ❖ Thiết lập các thuộc tính cho layout
- ❖ Thêm tất cả các nodes đã tạo vào trong layout



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

59

5. JavaFX - Layout Panes (Container)

- ❖ Sau khi tạo các node trong 1 scene, cần sắp xếp trình bày các node
- ❖ Layout của 1 container: là cách sắp xếp các node nằm trong container đó
- ❖ Các loại layout trong JavaFX: HBox, VBox, Border Pane, Stack Pane, Text Flow, Anchor Pane, Title Pane, Grid Pane, Flow Panel, ...
- ❖ Mỗi loại layout ứng với 1 class, tất cả các class này nằm trong gói javafx.layout, lớp Pane là lớp cơ sở của tất cả các lớp layout



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

58

Ví dụ với layout HBox

- ❖ Đặc điểm: các node xếp theo hàng ngang
- ❖ Một số thuộc tính quan trọng:
 - alignment: giống hàng các node
 - spacing: khoảng cách giữa các node
- ❖ Khởi tạo HBox

```
// Khởi tạo rỗng  
HBox hbox = new HBox();
```

```
// Khởi tạo với các node  
Button button1 = new Button("Button Number 1");  
Button button2 = new Button("Button Number 2");  
HBox hbox = new HBox(button1, button2);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

60

15

Ví dụ với layout HBox

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");
        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button Number 2");
        Button button3 = new Button("Button Number 3");
        HBox hbox = new HBox(button1, button2);
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.BOTTOM_CENTER);
        hbox.getChildren().add(button3);
        Scene scene = new Scene(hbox, 400, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

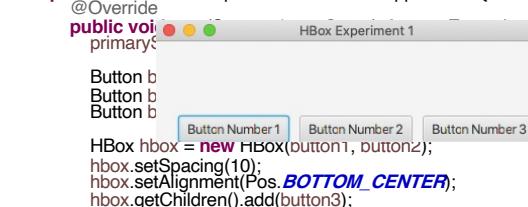


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

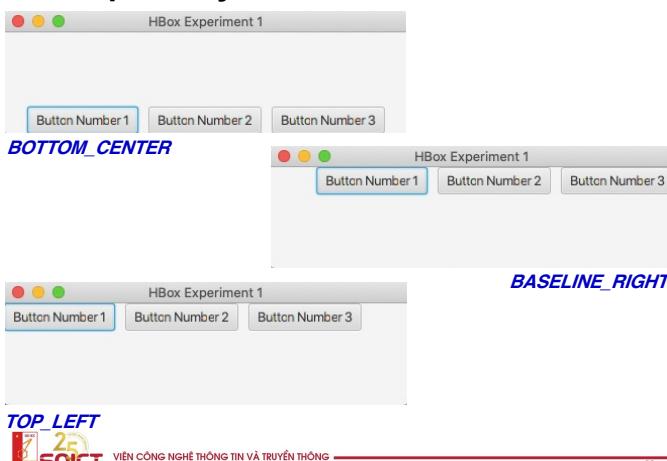
Ví dụ với layout HBox

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");
        Button b1 = new Button("Button Number 1");
        Button b2 = new Button("Button Number 2");
        Button b3 = new Button("Button Number 3");
        HBox hbox = new HBox(b1, b2);
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.BOTTOM_CENTER);
        hbox.getChildren().add(b3);
        Scene scene = new Scene(hbox, 400, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



62

Ví dụ với layout HBox



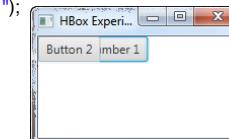
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

Ví dụ với layout Group

- ❖ Group: không sắp xếp các component trong nó, tất cả đều ở tọa độ (0, 0)

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class GroupExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");
        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button 2");
        Group group = new Group();
        group.getChildren().add(button1);
        group.getChildren().add(button2);
        Scene scene = new Scene(group, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



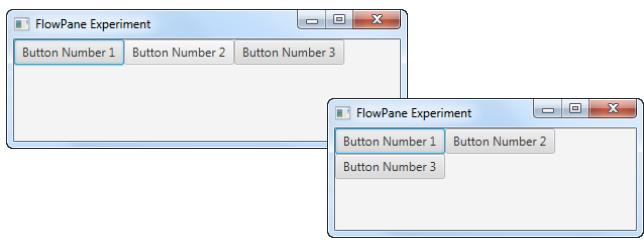
2 button đều ở tọa độ (0, 0), đè lên nhau

64

63

Một số layout khác

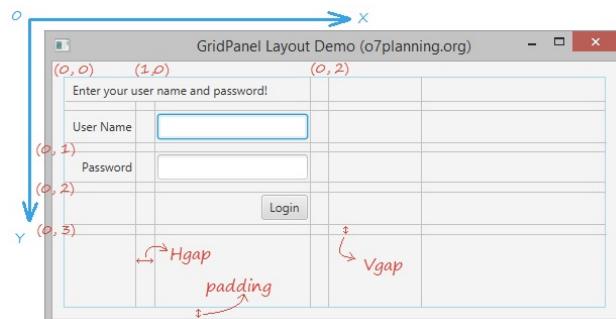
- ❖ FlowPane: sắp xếp các thành phần con liên tiếp nhau trên một dòng, và tự động đẩy phần tử con xuống dòng tiếp theo nếu dòng hiện tại không còn chỗ trống



65

Một số layout khác

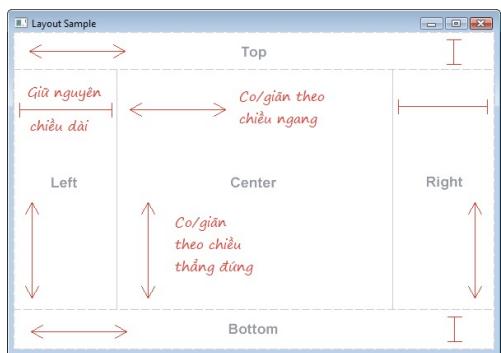
- ❖ GridPane: chia thành lưới gồm các hàng và các cột. Một thành phần con có thể nằm trên một ô lưới hoặc nằm trên một ô hợp nhất từ các ô gần nhau



66

Một số layout khác

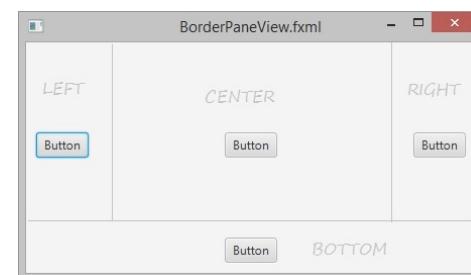
- ❖ BorderPane: chia thành 5 vùng riêng biệt, mỗi vùng có thể chứa được một thành phần con.



67

Một số layout khác

- ❖ BorderPane: Nếu một vùng nào đó không chứa thành phần con, các vùng khác sẽ chiếm lấy không gian của nó.
- ❖ Ví dụ: Vùng TOP không có thành phần con, không gian của nó sẽ bị các thành phần khác chiếm chỗ:



68

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. **Mô hình xử lý sự kiện**
7. Kéo thả giao diện với SceneBuilder



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

69

6. Mô hình xử lý sự kiện

- ❖ Lớp cơ sở cho các loại sự kiện: javafx.event.Event
- ❖ JavaFX hỗ trợ xử lý nhiều loại sự kiện
 - **Mouse Event** – Sự kiện xảy ra khi nhấn chuột (mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target). Lớp tương ứng là **MouseEvent**.
 - **Key Event** – Sự kiện xảy ra khi nhấn phím (key pressed, key released and key typed). Lớp tương ứng là **KeyEvent**.
 - **Drag Event** – Sự kiện xảy ra khi rê chuột (drag entered, drag dropped, drag entered target, drag exited target, drag over). Lớp tương ứng là **DragEvent**.
 - **Window Event** – Sự kiện xảy ra khi hiện/ẩn cửa sổ (window hiding, window shown, window hidden, window showing). Lớp tương ứng là **WindowEvent**.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

71

71

6. Mô hình xử lý sự kiện

- ❖ Các sự kiện được chia làm hai loại:

- **Foreground Events** – Là sự kiện cần người dùng tương tác trực tiếp. VD: nhấn chuột vào button, di chuyển chuột, gõ ký tự, chọn 1 item trong list, cuộn trang, ...
- **Background Events** – VD: can thiệp của hệ điều hành, lỗi phần mềm/phần cứng, hết giờ, hoàn thiện 1 thao tác gì đó, ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

70

6. Mô hình xử lý sự kiện

- ❖ Xử lý sự kiện (Event Handling): cài đặt code - sẽ được thực thi khi một sự kiện xác định nào đó xảy ra
- ❖ JavaFX cung cấp các handlers và các filters để xử lý sự kiện. Mỗi sự kiện sẽ có 3 thuộc tính:
 - **Event Target** – Node xảy ra sự kiện. Target có thể là stage, scene, hoặc một node
 - **Event Source** – Là đối tượng có trạng thái thay đổi, nó sinh ra sự kiện. Ví dụ: chuột, bàn phím, ...
 - **Event Type** – Kiểu của sự kiện. Ví dụ, với nguồn sự kiện là chuột, kiểu của sự kiện có thể là mouse pressed, mouse released
- ❖ Khi sự kiện xảy ra, event source tạo một đối tượng event và chuyển đối tượng này đến bộ xử lý sự kiện



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

72

18

Ví dụ

- ❖ Với ứng dụng JavaFX như trong hình, nếu nhấp chuột vào nút play, source sẽ là chuột, target là nút play, kiểu của sự kiện sinh ra là mouse click



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

73

Các bước xử lý sự kiện trong JavaFX

❖ 4 bước:

- Target selection
- Route construction
- Event capturing
- Event bubbling

<https://docs.oracle.com/javafx/2/events/processing.htm>

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

74

Các bước xử lý sự kiện trong JavaFX

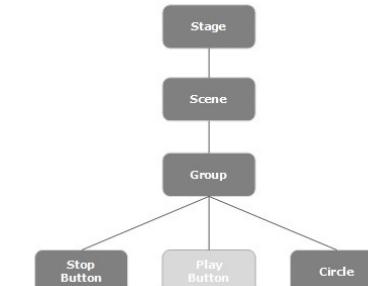
- ❖ Bước 1: Target selection (xác định **target node**). Khi một hành động xảy ra, hệ thống xác định **target node** theo các luật sau:
 - Với sự kiện nhấn phím, **target node** là node đang được focus
 - Với sự kiện nhấn chuột, **target node** là node ứng với vị trí hiện tại của chuột
 - ... (một số sự kiện khác trên thiết bị cảm ứng)

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

75

Các bước xử lý sự kiện trong JavaFX

- ❖ Bước 2: Route Construction – Tạo chuỗi sự kiện phát sinh (**Event Dispatch chain**): là đường đi từ stage tới **target node**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

76

Các bước xử lý sự kiện trong JavaFX

❖ Bước 3: Event Capturing (bắt sự kiện)

- Sau khi tạo chuỗi sự kiện, root node của ứng dụng sẽ gửi đi sự kiện (dispatch event).
- Sự kiện này sẽ đi dọc theo các node từ trên xuống dưới (top to bottom). Nếu một node nào đó đăng ký một **filter** cho sự kiện sinh ra, **filter** đó sẽ được thực thi.
- Nếu một filter nào đó **consume** event bằng cách gọi phương thức **consume()** từ đối tượng event tạo ra, quá trình xử lý sự kiện lập tức kết thúc
- Nếu event chưa được consume, cuối cùng sự kiện sẽ được chuyển tới cho **target node**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

77

77

Các bước xử lý sự kiện trong JavaFX

- ❖ Event Handlers và Event Filters: chứa logic ứng dụng để xử lý một sự kiện
- ❖ Một node có thể đăng ký nhiều handler/filter.
- ❖ filter/handler cho parent node có thể được cài đặt như xử lý mặc định cho tất cả các node con của nó
- ❖ Tất cả các handlers và filters đều thực thi giao diện **javafx.event.EventHandler**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

79

79

Các bước xử lý sự kiện trong JavaFX

❖ Bước 4: Nối bọt sự kiện (Event Bubbling)

- Sự kiện sẽ đi ngược lên trên, từ **target node** tới **root node** (bottom to top).
- Nếu bất kỳ một node nào đó trong **event dispatch chain** đăng ký một **handler** cho sự kiện sinh ra, **handler** sẽ được thực thi.
- Nếu không handler nào consume event, sự kiện sẽ chuyển tới root node, và hoàn thành việc xử lý



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

78

78

Thêm/bỏ filter

❖ Thêm filter

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>(){
    @Override
    public void handle(MouseEvent e) {
        System.out.println("Hello World");
        circle.setFill(Color.DARKSLATEBLUE);
    }
};

//Adding event Filter
Circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```

❖ Bỏ filter

```
circle.removeEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

80

80

20

Thêm/bỏ handler

❖ Thêm handler

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent e) {
        System.out.println("Hello World");
        circle.setFill(Color.DARKSLATEBLUE);
    }
};

//Adding event handler
Circle.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

❖ Bỏ handler

```
circle.removeEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

81

Ví dụ (1/3)

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class EventFiltersExample extends Application {
    @Override
    public void start(Stage stage) {
        Button button = new Button("Button");
        TextArea text = new TextArea();
        Circle circle = new Circle(25.0f);

        FlowPane fp = new FlowPane(button, text, circle);
        fp.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent arg0) {
                text.appendText("Filter in flow pane\n");
            }
        });
        fp.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent arg0) {
                text.appendText("Handler in flow pane\n");
            }
        });
    }
}
```

82

81

Ví dụ (2/3)

```
button.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        text.appendText("Filter in button\n");
    }
});
button.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        text.appendText("Handler in button\n");
    }
});

circle.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        text.appendText("Filter in circle\n");
    }
});
circle.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        text.appendText("Handler in circle\n");
    }
});

// Creating a scene object
Scene scene = new Scene(fp, 600, 300);
stage.setTitle("Event Filters Example");
stage.setScene(scene);
stage.show();
}
```

83

Ví dụ (3/3)

```
import javafx.application.Application;

public final class Main {
    public static void main(final String[] args) {
        Application.launch(EventFiltersExample.class, args);
    }
}
```



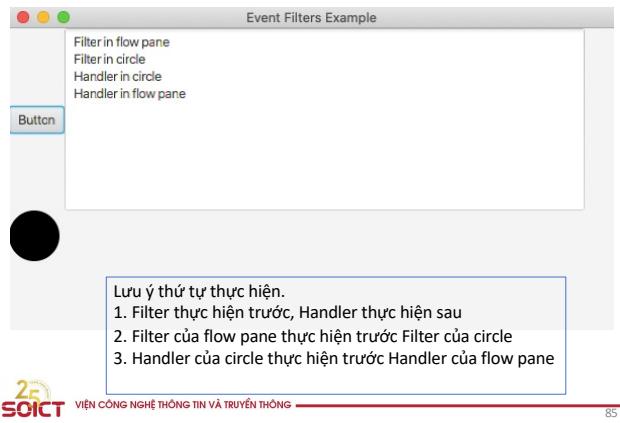
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

84

83

Ví dụ

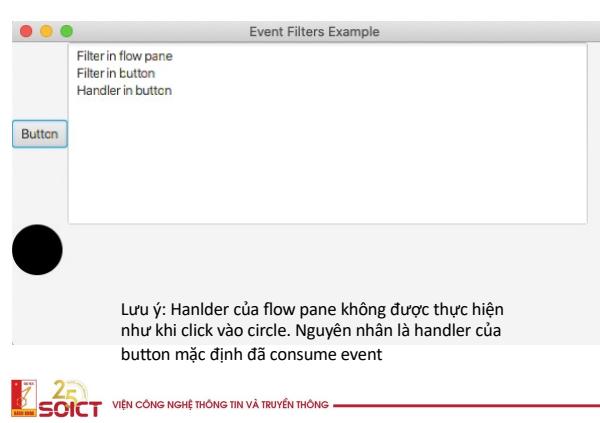
❖ Khi nhấn chuột vào circle



85

Ví dụ

❖ Khi nhấn chuột vào button



86

Ví dụ

❖ Sửa đổi lại filter của flow pane như sau

```
fp.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {  
    @Override  
    public void handle(MouseEvent arg0) {  
        text.appendText("Filter in flow pane\n");  
        arg0.consume();  
    }  
});
```

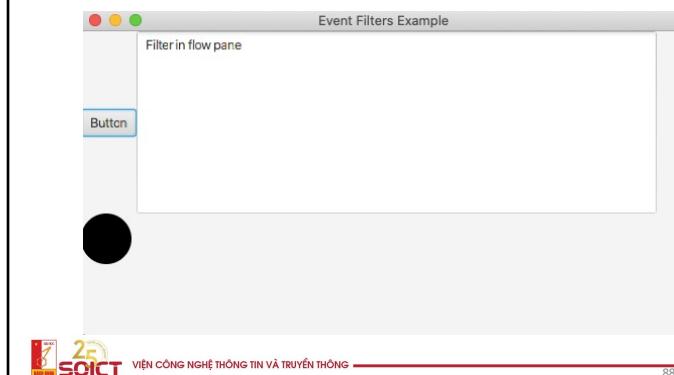


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

87

Ví dụ

❖ Khi nhấp chuột vào circle (hoặc button), do event đã được consume, nên kết quả như sau:



88

Nội dung

1. Giới thiệu
2. Cài đặt JavaFX
3. Các thành phần giao diện JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Mô hình xử lý sự kiện
7. **Kéo thả giao diện với SceneBuilder**



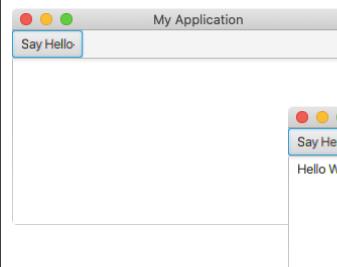
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

89

89

Ví dụ

- ❖ Viết ứng dụng: Khi nhấn button “Say Hello”, in dòng chữ Hello World ra textbox



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

91

91

7. Kéo thả giao diện với SceneBuilder

- ❖ Ý tưởng: tách biệt giao diện với logic xử lý nghiệp vụ
 - Giao diện ứng dụng: thiết kế trong file fxml
 - Logic xử lý (controller): tách biệt riêng trong file mã nguồn Java
- ❖ Các bước thực hiện:
 - Cài đặt SceneBuilder
 - Tạo giao diện (file fxml), định nghĩa các thuộc tính cho các component (tên component, các phương thức xử lý sự kiện)
 - Tạo JavaFX project
 - Copy file giao diện fxml vào JavaFX project
 - Cài đặt controller
 - Kết nối file giao diện fxml với controller
 - Tạo ứng dụng JavaFX, load file fxml

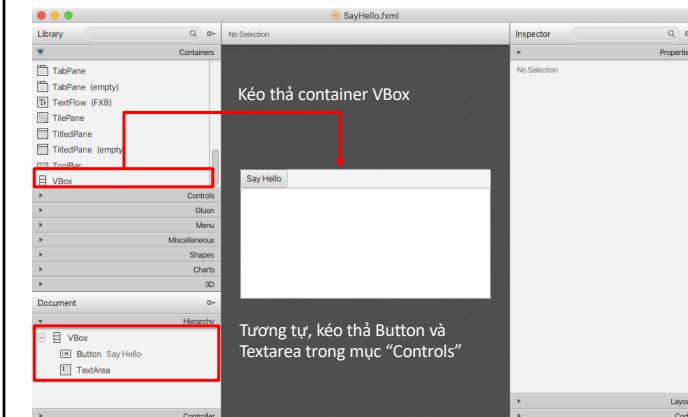


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

90

90

Tạo file SayHello.fxml với SceneBuilder

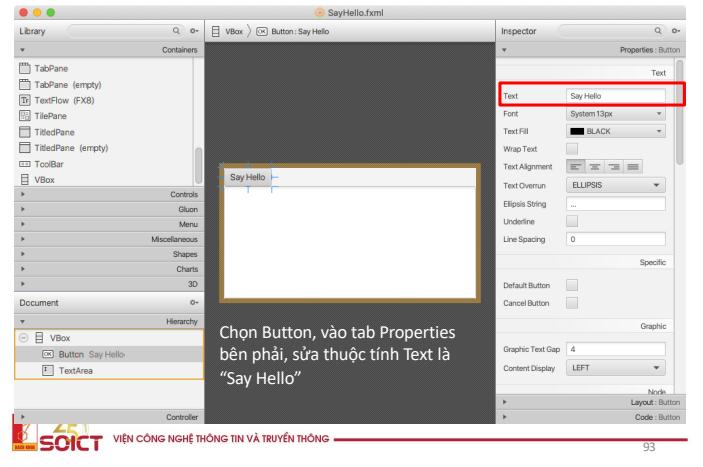


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

92

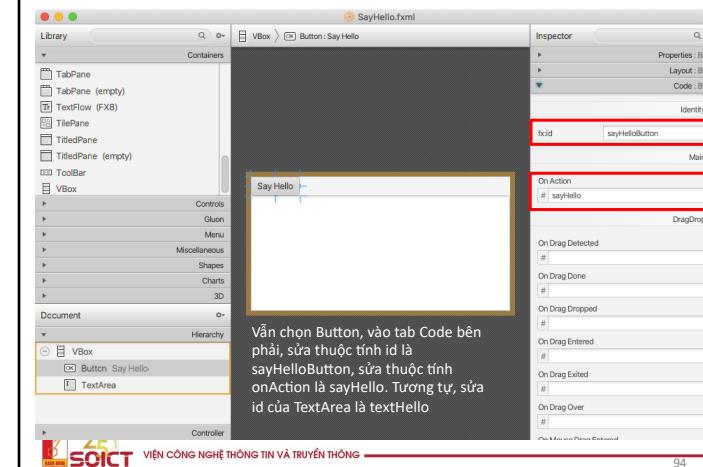
92

Chỉnh sửa thuộc tính của button



93

Chỉnh sửa thuộc tính của button



94

Tạo project JavaFX

- ❖ Tạo project JavaFX như bình thường, copy file SayHello.fxml vào project



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



95

Cài đặt Controller: tạo lớp MyController

```
import java.net.URL;
import java.util.ResourceBundle;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;

public class MyController implements Initializable {
    @FXML
    private Button sayHelloButton;

    @FXML
    private TextArea textHello;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

    public void sayHello(ActionEvent event) {
        textHello.setText("Hello World");
    }
}
```

Lưu ý: tên Button và tên TextArea phải khớp với các id tạo trong SceneBuilder

96

24

Kết nối file giao diện fxml với controller

- ❖ Sửa lại file SayHello.fxml: thêm thuộc tính fx:controller cho thẻ VBox, trả tới lớp MyController vừa tạo (dùng full name)

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.layout.VBox?>

<VBox prefHeight="192.0" prefWidth="371.0"
      xmlns="http://javafx.com/javafx/11.0.1"
      xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="oop.hust.MyController">
  <children>
    <Button fx:id="sayHelloButton" mnemonicParsing="false"
            onAction="#sayHello" text="Say Hello" />
    <TextArea fx:id="textHello" prefHeight="173.0" prefWidth="162.0" />
  </children>
</VBox>
```

Tạo ứng dụng JavaFX, load file fxml

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class MyApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            //Đọc file fxml và vẽ giao diện.
            Parent root = FXMLLoader.load(getClass()
                .getResource("/oop/hust/SayHello.fxml"));

            primaryStage.setTitle("My Application");
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Tài liệu tham khảo

- ❖ <http://tutorials.jenkov.com/javafx/overview.html>



Bài 12. Phân tích thiết kế hướng đối tượng và biểu đồ lớp

1

Nội dung

1. Phân tích thiết kế hướng đối tượng
2. Biểu đồ lớp
3. Quan hệ giữa các lớp
4. Ví dụ và bài tập

2

Nội dung

1. **Phân tích thiết kế hướng đối tượng**
2. Biểu đồ lớp
3. Quan hệ giữa các lớp
4. Ví dụ và bài tập



3

Tầm quan trọng của OOAD

- ❖ Nhiều người phát triển dự án
 - Cho rằng phần mềm chủ yếu được xây dựng bằng cách gõ "code" từ bàn phím
 - Không dành đủ thời gian cho quá trình phân tích và thiết kế phần mềm
- ❖ → Họ phải "cày bừa" để hoàn thành chương trình vì
 - Không hiểu hoặc hiểu sai yêu cầu
 - Giao tiếp với các thành viên không tốt
 - Không tích hợp được với module của đồng nghiệp...
- ❖ → Họ nhận ra rằng "Phân tích" và "Thiết kế" cần được coi trọng hơn, nhưng đã quá muộn

4

3

4

Tầm quan trọng của OOAD (2)

- ❖ Cần thiết lập một cơ chế hiệu quả để nắm bắt yêu cầu, phân tích thiết kế
- ❖ Cơ chế này phải như là một “ngôn ngữ thống nhất” giúp cho quá trình hợp tác hiệu quả giữa các thành viên trong nhóm phát triển phần mềm.
- ❖ → OOAD: Object Oriented Analysis and Design)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

5

Phương pháp OOAD

- ❖ OOAD được chia thành 2 giai đoạn
 - Phân tích hướng đối tượng (OOA)
 - Thiết kế hướng đối tượng (OOD)
- ❖ OOA là giai đoạn nhằm tạo ra các mô hình cơ bản (mô hình khái niệm) của hệ thống dựa theo những gì khách hàng yêu cầu về hệ thống của họ
- ❖ OOD sẽ bổ sung thêm các thông tin thiết kế chi tiết cho các mô hình nói trên



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

7

Mục đích của OOAD

- ❖ Chuyển các yêu cầu của bài toán thành một bản thiết kế của hệ thống sẽ được xây dựng
- ❖ Tập trung vào quá trình phân tích các YÊU CẦU của hệ thống và thiết kế các MÔ HÌNH cho hệ thống đó trước giai đoạn lập trình
- ❖ Được thực hiện nhằm đảm bảo mục đích và yêu cầu của hệ thống được ghi lại một cách hợp lý trước khi hệ thống được xây dựng
- ❖ Cung cấp cho người dùng, khách hàng, kỹ sư phân tích, thiết kế nhiều cái nhìn khác nhau về cùng một hệ thống

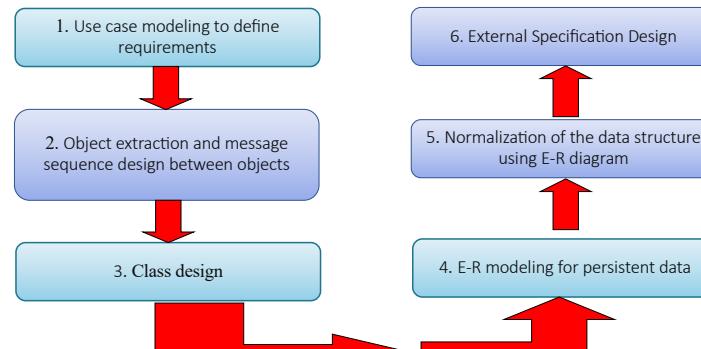


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

6

Phương pháp OOAD (2)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

8

OOA

- ❖ Xác định yêu cầu phần mềm
- ❖ ĐẶc tả yêu cầu phần mềm thông qua mô hình các đối tượng và tương tác giữa chúng
- ❖ Tạo được mô hình có các thành phần là đối tượng và khái niệm đời thực, dễ hiểu với người dùng
- ❖ Mô hình hóa các thực thể, giữ nguyên cấu trúc, quan hệ, hành vi giữa chúng

OOD

- ❖ Thực thi các mô hình khái niệm là đầu ra của bước OOA
- ❖ Các khái niệm trong OOA được ánh xạ theo thành các lớp thực thi. Các ràng buộc, các giao diện được thiết kế. Kết quả là đặc tả chi tiết về hệ thống cần xây dựng, theo một công nghệ cụ thể được lựa chọn

OOA (2)

- ❖ Ví dụ với 1 phòng bán ô tô:
 - Các thực thể:
 - Khách hàng
 - Người bán hàng
 - Phiếu đặt hàng
 - Phiếu (hoá đơn) thanh toán
 - Xe ô tô
 - Tương tác và quan hệ giữa các thực thể trên :
 - Người bán hàng dẫn khách hàng tham quan phòng trưng bày xe.
 - Khách hàng chọn một chiếc xe
 - Khách hàng viết phiếu đặt xe
 - Khách hàng trả tiền xe
 - Xe ô tô được giao đến cho khách hàng

OOD

- ❖ Tổ chức chương trình thành các tập hợp đối tượng cộng tác
 - Mỗi đối tượng là thực thể của một lớp
- ❖ Thiết kế trên kết quả của OOA
 - Cải thiện, tối ưu hóa thêm
 - Thiết kế các
 - Phương thức (operations)
 - Thuộc tính (attributes)
 - Mối quan hệ giữa các lớp (classes)
 - Đưa ra các biểu đồ tĩnh và động
 - Tĩnh: biểu thị các lớp và đối tượng
 - Động: biểu thị tương tác giữa các lớp & phương thức hoạt động

Thiết kế biểu đồ lớp

- Mục tiêu: cần xác định các thành viên của mỗi lớp và quan hệ giữa các lớp
- Một trong các kỹ thuật được ứng dụng nhiều nhất là Thẻ Class-Responsibility-Collaboration (CRC) card.
- Mỗi thẻ thể hiện một lớp, trên thẻ chúng ta lưu lại các thông tin sau về các lớp:
 - Tên của lớp. Thông thường người ta đặt tên lớp liên quan đến vai trò của lớp, chúng ta sẽ sử dụng lớp để làm gì.
 - Trách nhiệm của lớp: lớp có thể làm gì. Thông thường các thông tin ở đây bao gồm tên của các hàm thành phần
 - Tương tác của lớp: lớp này có thể tương tác được với những lớp nào khác

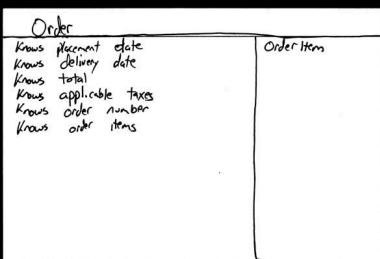
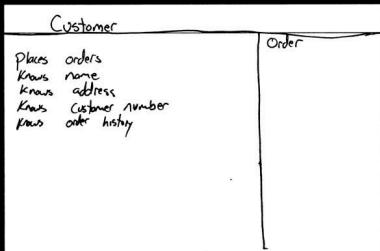


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

CRC Card

Class Name	
Responsibilities	Collaborators



14

Thiết kế đối tượng (1/2)

- Trong PT&TK hướng đối tượng người ta đã tổng kết 5 bước để thiết kế đối tượng:
 - Bước 1. Phát hiện đối tượng (Object discovery). Bước này được thực hiện ở giai đoạn phân tích chuỗi.
 - Bước 2. Lắp ráp đối tượng (Object assembly). Bước tìm kiếm các đặc điểm của đối tượng để thêm vào các thuộc tính, các hàm thành phần cho đối tượng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

Thiết kế đối tượng (2/2)

- Trong PT&TK hướng đối tượng người ta đã tổng kết 5 bước để thiết kế đối tượng:
 - Bước 3. Xây dựng hệ thống (System construction). Trong giai đoạn này chúng ta phát triển các đối tượng, xem xét các tương tác giữa các đối tượng để hình thành hệ thống hoạt động.
 - Bước 4. Mở rộng hệ thống (System extension). Khi chúng ta thêm vào các tính năng của hệ thống, cần thêm các lớp mới, các đối tượng mới và các tương tác giữa các đối tượng này với các đối tượng đã có trong hệ thống.
 - Bước 5. Tái sử dụng đối tượng (Object reuse). Đây là một trong những thử nghiệm quan trọng của các đối tượng và lớp trong thiết kế phần mềm. Chúng ta cần phải sử dụng lại các lớp và các đối tượng trong phần mềm (qua tính kế thừa và tương tác giữa các đối tượng)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

15

Lưu ý (1/2)

- ❖ Một số điểm lưu ý khi phát triển các lớp
 - 1. Cần tạo ra lớp trước, sau đó mới nghĩ tới việc phát triển và hoàn thiện lớp trong quá trình giải quyết bài toán
 - 2. Khi phân tích hay phát triển các lớp không nên tập trung xác định tất cả thành viên một lớp, chúng ta sẽ biết rõ hơn khi phát triển hệ thống (*learns as you go*)
 - 3. Việc phát hiện ra các lớp cần thiết cho chương trình là một trong những nhiệm vụ chính của thiết kế hệ thống, nếu chúng ta đã có những lớp này (trong một thư viện lớp nào đó chẳng hạn) thì công việc sẽ dễ dàng hơn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

Nội dung

1. Phân tích thiết kế hướng đối tượng
2. Biểu đồ lớp
3. Quan hệ giữa các lớp
4. Ví dụ và bài tập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

19

Lưu ý (2/2)

- ❖ Một số điểm lưu ý khi phát triển các lớp
 - 4. Khi lập trình cần tuân thủ theo các thiết kế đã làm. Không nên băn khoăn khi không sử dụng phương pháp lập trình truyền thống và thấy choáng ngợp trước số lượng lớn các đối tượng.
 - 5. Luôn giữ nguyên tắc: mọi vấn đề cần giải quyết theo phương án đơn giản nhất, không phức tạp hóa. Sử dụng nguyên lý của Occam Razor: *Lớp đơn giản nhất bao giờ cũng là lớp tốt nhất, hãy bắt đầu bằng những cái đơn giản và chúng ta sẽ kết thúc bằng những hệ thống phức tạp*



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

18

Lớp (Class)

- ❖ Sử dụng hình chữ nhật gồm 3 thành phần
 - Tên lớp
 - Các thuộc tính
 - Các phương thức

Class_Name
attribute1
attribute2
attribute3
method1()
method2()
method3()



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

20

Biểu diễn thuộc tính

- ❖ Chỉ ra tên, kiểu và giá trị mặc định nếu có
 - `attributeName : Type = Default`
- ❖ Tuân theo quy ước đặt tên của ngôn ngữ cài đặt và của dự án.
- ❖ Kiểu (type) nên là kiểu dữ liệu cơ bản trong ngôn ngữ thực thi
 - Kiểu dữ liệu có sẵn, kiểu dữ liệu người dùng định nghĩa, hoặc lớp tự định nghĩa.



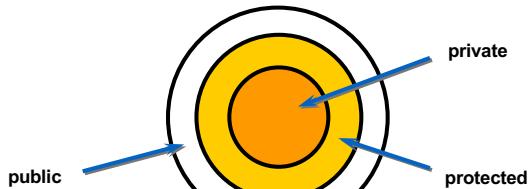
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

Phạm vi truy cập (Visibility)

- ❖ Phạm vi truy cập được sử dụng để thực hiện khả năng đóng gói



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

Mô tả phương thức

- ❖ Tên phương thức:
 - Mô tả kết quả
 - Sử dụng góc nhìn của đối tượng khách (client – đối tượng gọi)
 - Nhận quan giữa các lớp
- ❖ Chữ ký của phương thức:
 - `operationName([direction] parameter: class, ...): returnType`
 - Direction: `in` (mặc định), `out` hoặc `inout`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

Phạm vi truy cập được biểu diễn như thế nào?

- ❖ Các ký hiệu sau được sử dụng:
 - + Public access
 - # Protected access
 - - Private access

Class1
- privateAttribute
+ publicAttribute
protectedAttribute
- privateOperation ()
+ publicOperation ()
protectedOperation ()



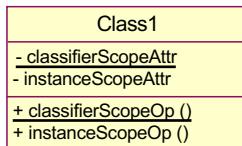
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

24

Phạm vi (Scope)

- ❖ Xác định số lượng thể hiện của thuộc tính/thao tác:
 - Instance: Một thể hiện cho mỗi thể hiện của mỗi lớp
 - Classifier: Một thể hiện cho tất cả các thể hiện của lớp
- ❖ Phạm vi Classifier được ký hiệu bằng cách gạch dưới tên thuộc tính/thao tác.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

25

Biểu đồ lớp là gì?

- ❖ Biểu đồ lớp chỉ ra sự tồn tại của các lớp và mối quan hệ giữa chúng trong bản thiết kế logic của một hệ thống
 - Chỉ ra cấu trúc tĩnh của mô hình như lớp, cấu trúc bên trong của chúng và mối quan hệ với các lớp khác.
 - Chỉ ra tất cả hoặc một phần cấu trúc lớp của một hệ thống.
 - Không đưa ra các thông tin tạm thời.
- ❖ Khung nhìn tĩnh của một hệ thống chủ yếu hỗ trợ các yêu cầu chức năng của hệ thống.

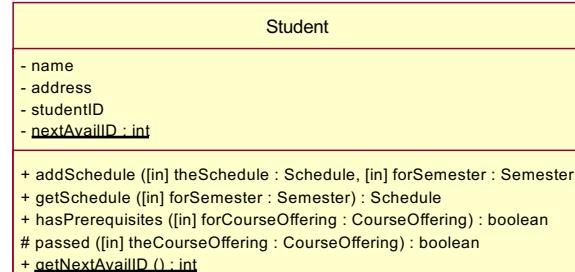


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

27

Ví dụ: Scope



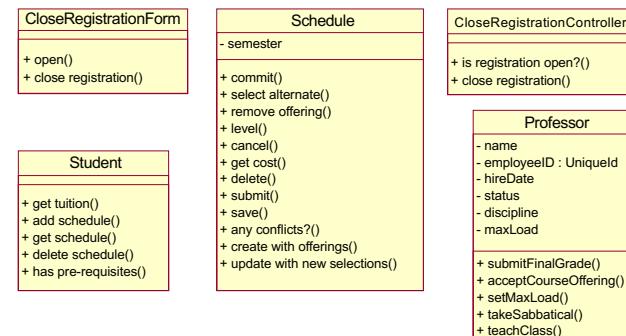
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

26

Biểu đồ lớp (Class Diagram – CD)

- ❖ Khung nhìn tĩnh của hệ thống



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

28

Khi nào sử dụng biểu đồ lớp?

- ❖ Từ vựng của hệ thống (Vocabulary)
 - Khi trừu tượng hóa một phần hoặc bên ngoài hoặc biên của hệ thống.
 - Chỉ ra kết quả trừu tượng hóa và trách nhiệm của chúng
- ❖ Cộng tác (Collaboration)
 - Nhóm các lớp và các thành phần khác làm việc cùng nhau để thực hiện một công việc nào đó.
- ❖ Lược đồ CSDL logic (Logical database schema)
 - Tương tự như bản thiết kế khái niệm cho CSDL
 - Chứa các đối tượng cần lưu trữ lâu dài tức là cần lưu trong CSDL

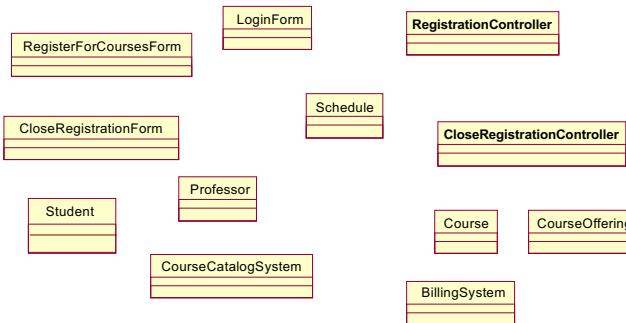


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

Ví dụ Biểu đồ lớp

- ❖ Có cách nào tốt hơn để tổ chức biểu đồ lớp?

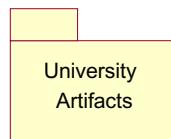


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

Gói (package)

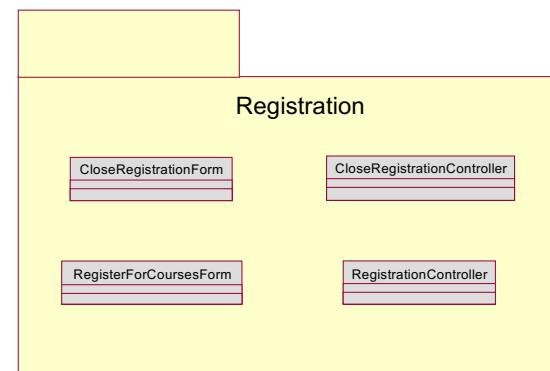
- ❖ Một cơ chế chung để tổ chức các phần tử thành nhóm.
- ❖ Một phần tử trong mô hình có thể chứa các phần tử khác.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

Ví dụ: Registration Package



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

Nội dung

1. Phân tích thiết kế hướng đối tượng
2. Biểu đồ lớp
3. **Quan hệ giữa các lớp**
4. Ví dụ và bài tập



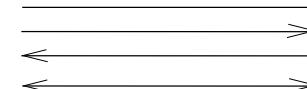
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

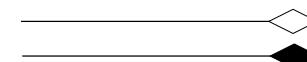
33

Class Relationships

❖ Association



- Aggregation
- Composition



❖ Inheritance



❖ Dependency



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

34

Liên kết (association) là gì?

- ❖ Mọi liên hệ ngữ nghĩa giữa hai hay nhiều lớp chỉ ra sự liên kết giữa các thể hiện của chúng
- ❖ Mọi quan hệ về mặt cấu trúc chỉ ra các đối tượng của lớp này có kết nối với các đối tượng của lớp khác.

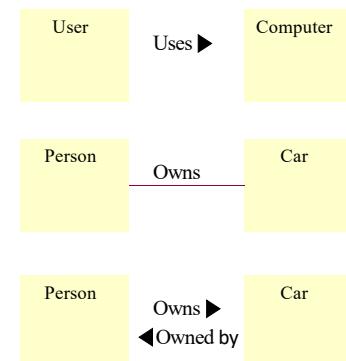


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

35

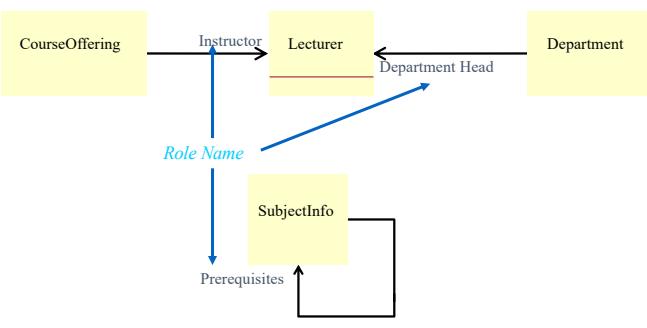
Tên và hướng của liên kết



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

Vai trò (role) trong liên kết



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

Biểu diễn bội số quan hệ

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

Bội số quan hệ (Multiplicity)

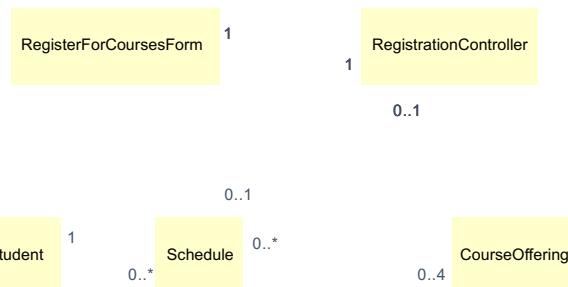
- ❖ Bội số quan hệ là số lượng thể hiện của một lớp liên quan tới MỘT thể hiện của lớp khác.
- ❖ Với mỗi liên kết, có hai bội số quan hệ cho hai đầu của liên kết.
 - Với mỗi đối tượng của Professor, có nhiều Course Offerings có thể được dạy.
 - Với mỗi đối tượng của Course Offering, có thể có 1 hoặc 0 Professor giảng dạy.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

Ví dụ về bội số quan hệ



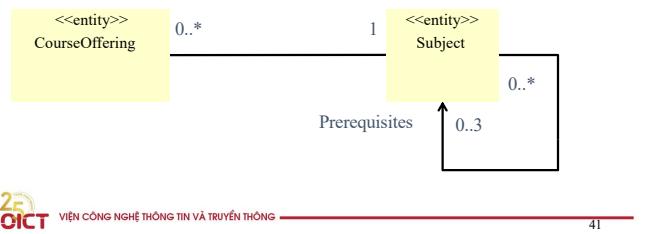
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

Ý nghĩa của bội số quan hệ

❖ Giúp trả lời 2 câu hỏi

- Liên kết là bắt buộc hay tùy chọn?
- Số lượng nhỏ nhất và lớn nhất các thể hiện của một lớp được liên kết với **một** thể hiện của lớp khác



41

Các loại liên kết

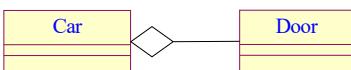
❖ Association

- use-a
- Các đối tượng của một lớp liên kết với các đối tượng của lớp khác



❖ Aggregation

- has-a/is-a-part
- Liên kết mạnh-Strong association. Thể hiện của một lớp được tạo bởi (**made up**) các thể hiện của lớp khác



❖ Composition

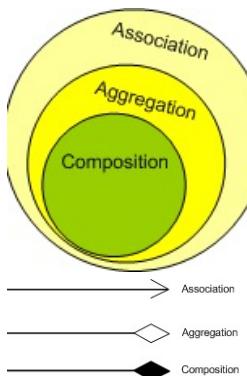
- Kết tập mạnh-Strong aggregation. Đối tượng bộ phận không thể được chia sẻ, và cùng bị hủy với đối tượng tổng thể
- Share life-time



42

Association, Aggregation and Composition

- ❖ Liên kết (Association)
 - Sử dụng (use-a)
- ❖ Kết tập (Aggregation)
 - Strong association
 - has-a/is-a-part
- ❖ Hợp thành (Composition)
 - Strong aggregation
 - Share life-time



43

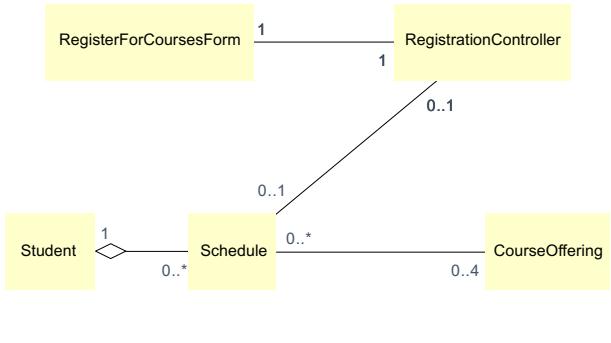
Kết tập (aggregation) là gì?

- ❖ Là một dạng đặc biệt của liên kết mô hình hóa mối quan hệ toàn thể-bộ phận (whole-part) giữa đối tượng toàn thể và các bộ phận của nó.
 - Kết tập là mối quan hệ “là một phần” (“is a part-of”).
- ❖ Bội số quan hệ được biểu diễn giống như các liên kết khác



44

Ví dụ về kết tập

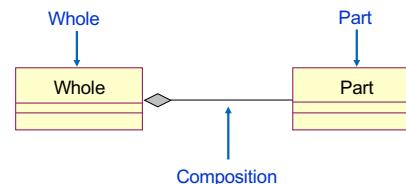


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

Hợp thành/Cấu thành (Composition) là gì?

- ❖ Một dạng của kết tập với quyền sở hữu mạnh và các vòng đời trùng khớp giữa hai lớp
 - Whole sở hữu Part, tạo và hủy Part.
 - Part bị bỏ đi khi Whole bị bỏ, Part không thể tồn tại nếu Whole không tồn tại.

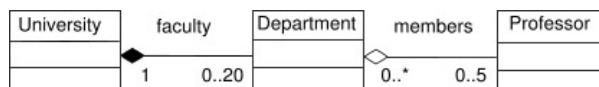


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

Ví dụ – Aggregation vs. Composition

- ❖ University chứa (own) nhiều Department
- ❖ Mỗi Department có 1 số các Professor



Nếu hủy University:

- Các phòng ban cũng không còn tồn tại
- Nhưng các Professor trong các Department vẫn còn tồn tại

Dấu hiệu khác:

- 1 professor có thể làm trong nhiều Department
- 1 Department chỉ thuộc về 1 University



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

Ví dụ Composition



Folder could contain many files, while each File has exactly one Folder parent. If Folder is deleted, all contained Files are deleted as well.

Hospital has 1 or more Departments, and each Department belongs to exactly one Hospital. If Hospital is closed, so are all of its Departments.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

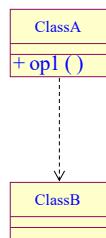
Phụ thuộc - Dependency

- ❖ Là quan hệ giữa 2 đối tượng của 2 lớp



Local Variable Visibility

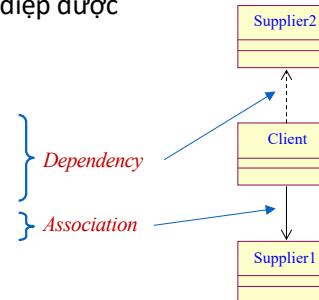
- ❖ Phương thức op1() tạo và sử dụng biến cục bộ tham chiếu tới đối tượng ClassB



Dependencies vs. Associations

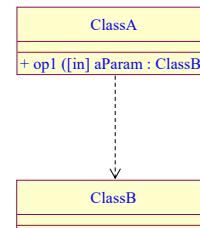
- ❖ Các đối tượng cần phải “biết nhau” để truyền thông điệp được cho nhau

- Local variable reference
- Parameter reference
- Global reference
- Field reference



Parameter Visibility

- ❖ Tham chiếu tới đối tượng lớp ClassB được truyền làm tham số trong phương thức op1 của lớp ClassA



Tổng quát hóa (Generalization)

- ❖ Mỗi quan hệ giữa các lớp trong đó một lớp chia sẻ cấu trúc và/hoặc hành vi với một hoặc nhiều lớp khác
- ❖ Xác định sự phân cấp về mức độ trừu tượng hóa trong đó lớp con kế thừa từ một hoặc nhiều lớp cha
 - Đơn kế thừa (Single inheritance)
 - Đa kế thừa (Multiple inheritance)
- ❖ Là mối liên hệ “là một loại” (“is a kind of”)



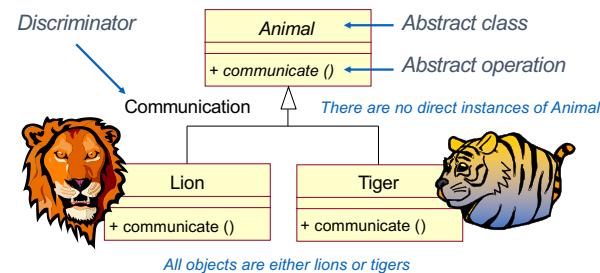
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

Lớp trừu tượng và lớp cụ thể (Abstract and Concrete Class)

- ❖ Lớp trừu tượng không thể có đối tượng
 - Chứa phương thức trừu tượng
 - Chữ nghiêng
- ❖ Lớp cụ thể có thể có đối tượng



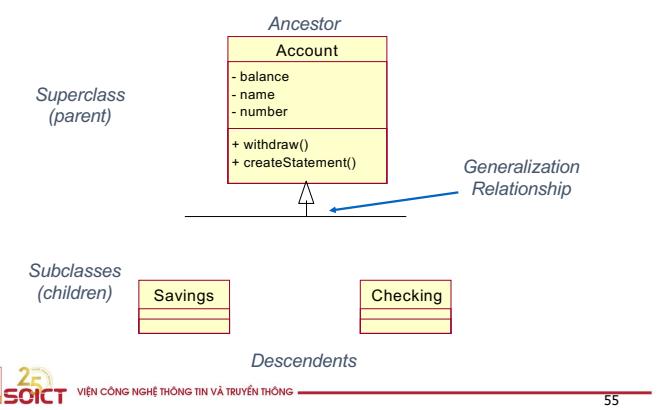
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

54

Ví dụ về Đơn kế thừa

- ❖ Một lớp kế thừa từ MỘT lớp khác



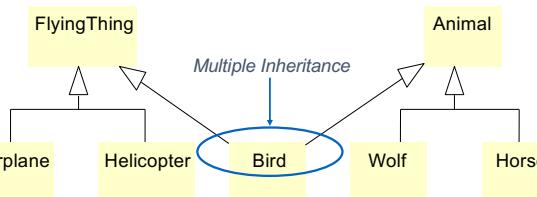
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

55

Ví dụ về Đa kế thừa

- ❖ Một lớp có thể kế thừa từ nhiều lớp khác



Sử dụng đa kế thừa chỉ khi cần thiết và luôn luôn phải cẩn thận!



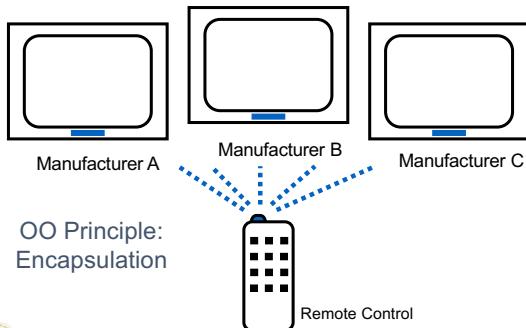
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

14

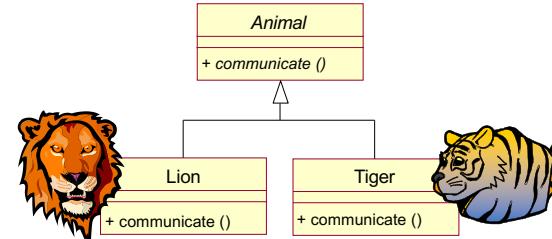
Đa hình (Polymorphism) là gì?

- ❖ Khả năng che giấu các thực thi khác nhau dưới một giao diện duy nhất.



57

Tổng quát hóa: Thực thi đa hình



Without Polymorphism

```
if animal = "Lion" then
    Lion communicate
else if animal = "Tiger" then
    Tiger communicate
end
```

With Polymorphism

```
Animal communicate
```

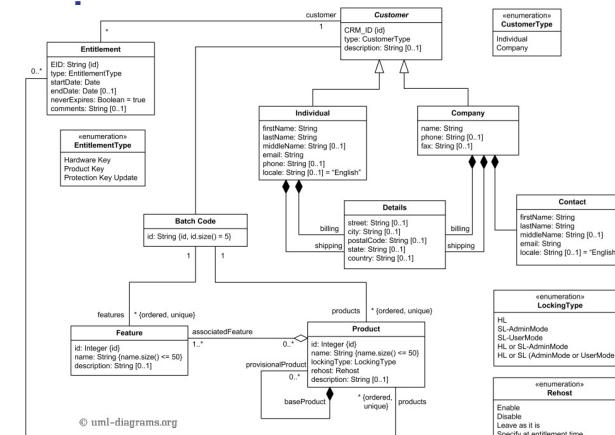
58

Nội dung

1. Phân tích thiết kế hướng đối tượng
2. Biểu đồ lớp
3. Quan hệ giữa các lớp
- 4. Ví dụ và bài tập**

59

Ví dụ



<https://www.uml-diagrams.org/software-licensing-domain-diagram-example.html> 60

60

Bài tập 1

- **Xây dựng phần mềm quản lý đăng ký dạy và học ở trường đại học như sau:**
- Các sinh viên và giảng viên được lưu trữ thông tin vào trong phần mềm này với các nội dung về tên, tuổi, định danh cá nhân, mã số sinh viên hoặc mã cán bộ. Giảng viên còn có thông tin về cấp bậc (**level** với dạng số nguyên từ 1 trở đi); và mã số thuế (**tax**).
- Mỗi giảng viên có thể dạy nhiều lớp (**Course**) và có thể chủ nhiệm nhiều sinh viên. Mỗi sinh viên cũng đăng ký nhiều lớp học (**Course**), ứng với mỗi sinh viên có một bảng điểm (**Table**) và mỗi sinh viên được chủ nhiệm bởi một giảng viên.
- Bảng điểm của một sinh viên lưu trữ thông tin điểm của từng lớp học của sinh viên đăng ký.
- Hãy xây dựng biểu đồ lớp

Bài tập 2

Bài tập 2

- **Chương trình quản lý Thông tin trong một nhà ga được mô tả như sau:**
- Hoạt động chuyên chở trong nhà ga gồm nhiều đoàn tàu. Mỗi đoàn tàu có một số hiệu riêng, thông tin về ga đích đến của đoàn tàu và lịch trình chạy của đoàn tàu (giờ khởi hành và giờ dự kiến đến ga đích).
- Một đoàn tàu gồm nhiều toa tàu. Mỗi toa thuộc một trong hai loại toa chở khách hoặc toa chở hàng. Mỗi toa tàu có một số hiệu duy nhất và trong lượng không tái tính bằng tấn (khi không chở khách hay hàng hóa). Mỗi toa chở khách còn có thông tin riêng về số lượng khách tối đa có thể chở.
- Khi tàu vào ga toa chở khách có thêm các hoạt động: thêm khách lên toa, bớt khách xuống toa.
- Thông tin về hành khách đi tàu gồm có họ tên, số chứng minh nhân dân, đoàn tàu và toa tàu mà họ mua vé. ga lên tàu và điểm xuống.
- Hãy xây dựng biểu đồ lớp

Bài tập 3

- **Một phần mềm Quản lý xe buýt tại bến xe được mô tả như sau:**
- Một xe buýt (**Bus**) chạy được tối đa 30 chuyến/ngày (**Trip**). Mỗi chuyến chứa tối đa 80 hành khách (**Person**). Hành khách được chia làm hai loại: hành khách mua vé theo từng lượt đi (**Customer**) và hành khách mua vé tháng (**Passenger**).
- Tất cả các hành khách đều được định danh bằng tên (**name**) và số chứng minh thư (**citizenCard**). Khách mua vé tháng có thêm thông tin mã vé ID.
- Các xe buýt có thông số về số lượng ghế ngồi (**numberOfSeats**) khác nhau. Trong lớp **Bus**, người ta xây dựng phương thức **public isEnableToLeaveStation(Trip t)**, trả về true nếu số hành khách trên chuyến xe buýt t bé hơn hoặc bằng 80% số ghế ngồi. Lớp **Bus** là lớp toàn thể, lớp **Trip** kết tập trong nó với tên vai trò là **trips**.
- Các xe buýt có thông số để định danh, đây là một con số. Trong lớp **Trip**, người ta xây dựng phương thức **public availableSeats()** trả về số lượng các ghế trống có trên chuyến xe.
- Người ta cài đặt trong lớp **Trip** phương thức mang tên **numberOfPassenger()**, trả về số lượng các khách sử dụng vé tháng có trên chuyến xe.
- Hãy xây dựng biểu đồ lớp