LỜI NÓI ĐẦU

Ngôn ngữ lập trình java ra đời và được các nhà nghiên cứu của Công ty Sun Microsystem giới thiệu vào năm 1995. Sau khi ra đời không lâu, ngôn ngữ lập trình này đã được sử dụng rộng rãi và phổ biến đối với các lập trình viên chuyên nghiệp cũng như các nhà phát triển phần mềm. Gần đây ngôn ngữ lập trình, công nghệ java đã được đưa vào giảng dạy ở các cơ sở đào tạo lập trình viên chuyên nghiệp. Một số trường đại học ở Việt Nam dạy môn lập trình java như một chuyên đề tự chọn cho các sinh viên công nghệ thông tin giai đoạn chuyên ngành.

Bố cục của bài gồm 6 phần:

Phần 1: GIỚI THIỆU TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

Phần 2: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH JAVA

Phần 3: HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Phần 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

Phần 5: LẬP TRÌNH CƠ SỞ DỮ LIỆU

Phần 6: PHẦN MỀM ỨNG DỤNG

CHƯƠNG I
GIỚI THIỆU TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

1.1. Lịch sử java.

- Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995.
- Ban đầu Gosling sử dụng C++ để viết phần mềm điều khiển , hiển thị số cho thiết bị như: VCR(video cassette recoder). PDA(personal digital assitant) .
- Gosling đã tạo ra một ngôn ngữ lập trình mới có tên là Oka. Ngôn ngữ này có cú pháp như C++ , nhưng bỏ qua những tính năng nguy hiểm của C++.
- Java được công bố chính thức vào năm 1995 và ngay lập tức đã tạo nên một trào lưu mới trên toàn thế giới và từ đó đến nay vẫn được có sức cuốn hút mạnh mẽ. Bởi vì Java không chỉ là một ngôn ngữ lập trình mà nó là giải pháp cho nhiều vấn đề.

1.2. Java là gì?

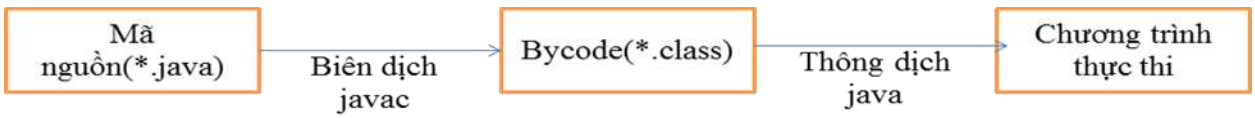
- Java công nghệ cho phép tạo ra các phần mềm phân tán (distributed sofware). Đây là những phần mềm đặt trên máy chủ (server) được nạp về qua kết nối mạng và thực hiện trên máy khách (client).

-Chương trình viết bằng ngôn ngữ lập trình java có thể chạy trên bất kỳ hệ thống nào có cài máy ảo java (Java Virtual Machine).

- Là ngôn ngữ lập trình hướng đối tượng
- Java là ngôn ngữ vừa biên dịch vừa thông dịch
- Cho phép viết các chương trình mạnh và tin cậy
- Xây dựng ứng dụng chạy được trên hầu hết các phần cứng và hệ điều hành khác nhau(multi-platform)
- Phân phối các ứng dụng trên mạng với độ bảo mật và an toàn cao.

1.3. Cấu trúc java

Chạy một chương trình viết bằng java:



- Biên dịch: Chương trình nguồn viết bằng ngôn ngữ java có đuôi (.java),sau khi được biển dịch.
- Bằng trình biên dịch javac sẽ chuyển chương trình nguồn java thành các byte code nằm trong tập tin *.class. Các byte code này chỉ có thể chạy trên máy ảo java (java virtual machine –JVM)
- Thông dịch : Khi thực hiện chương trình java ,máy ảo java sử dụng trình lạp lớp (class loader) để đọc các byte code từ đĩa hoặc kết nối mạng. Các lớp được nạp sẽ phải đi qua trình kiểm tra lớp(class verifler) để chắc chắn rằng chúng không sinh ra các lỗi ảnh hưởng đến hệ thống khi thực thi.
- Phần thực hiện (excution unit) trong máy ảo java sẽ thực thi các lệnh quy định trong từng byte code.

- Cấu trúc máy ảo java:
  + Chương trình java: Tập hợp các đối tượng
  + Máy ảo java
  + Hệ điều hành
- Máy ảo java bao gồm các phần:
  + Trình nạp lớp(class loader)
  + Trình kiểm tra lớp(class verifier)
  + Trình thực thi (excution unit)
- Bộ java Deverlopers Kit (JDK) do Sun Microsystems cung cấp trình tiện ích cho phép biên dịch, bắt lỗi và tạo tài liệu cho một ứng dụng java:
  + Javac: Bộ biên dịch java - làm nhiệm vụ chuyển mã nguồn java sang byte code
  + Java : Bộ thông dịch java -thực thi các ứng dụng java trực tiếp từ tập tin lớp (class)
  + Appletviewer: Một trình thông dịch java thực thi các java từ tập tin HTML.

1.4. Một số đặc điểm nổi bậc của ngôn ngữ lập trình Java Máy ảo Java (JVM - Java Virtual Machine)

- Tất cả các chương trình muốn thực thi được thì phải được biên dịch ra mã máy. Mã máy của từng kiến trúc CPU của mỗi máy tính là khác nhau (tập lệnh mã máy của CPU Intel, CPU Solarix, CPU Macintosh … là khác nhau), vì vậy trước đây một chương trình sau khi được biên dịch xong chỉ có thể chạy được trên một kiến trúc CPU cụ thể nào đó. Đối với CPU Intel chúng ta có thể chạy các hệ điều hành như Microsoft Windows, Unix, Linux, OS/2, … Chương trình thực thi được trên Windows được biên dịch dưới dạng file có đuôi .EXE còn trên Linux thì được biên dịch dưới dạng file có đuôi .ELF, vì vậy trước đây một chương trình chạy được trên Windows muốn chạy được trên hệ điều hành khác như Linux chẳng hạn thì phải chỉnh sửa và biên dịch lại. Ngôn ngữ lập trình Java ra đời, nhờ vào máy ảo Java mà khó khăn nêu trên đã được khắc phục. Một chương trình viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo java (mã java bytecode). Sau đó máy ảo Java chịu trách nhiệm chuyển mã java bytecode thành mã máy tương ứng. Sun Microsystem chịu trách nhiệm phát triển các máy ảo Java chạy trên các hệ điều hành trên các kiến trúc CPU khác nhau.

- **Thông dịch:** Java là một ngôn ngữ lập trình vừa biên dịch vừa thông dịch. Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi *.java đầu tiên được biên dịch thành tập tin có đuôi *.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.
- **Độc lập nền:** Một chương trình viết bằng ngôn ngữ Java có thể chạy trên nhiều máy tính có hệ điều hành khác nhau (Windows, Unix, Linux, …) miễn sao ở đó có cài đặt máy ảo java (Java Virtual Machine). Viết một lần chạy mọi nơi ***(write once run anywhere).***

- **Hướng đối tượng:** Hướng đối tượng trong Java tương tự như C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn. Tất cả mọi thứ đề cập đến trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính của một chương trình viết bằng Java (đó là hàm main) cũng phải đặt bên trong một lớp. Hướng đối tượng trong Java không có tính đa kế thừa (multi inheritance) như trong C++ mà thay vào đó Java đưa ra khái niệm interface để hỗ trợ tính đa kế thừa.
- **Đa nhiệm - đa luồng (MultiTasking - Multithreading):** Java hỗ trợ lập trình đa nhiệm, đa luồng cho phép nhiều tiến trình, tiểu trình có thể chạy song song cùng một thời điểm và tương tác với nhau.

  Khả chuyển (portable): Chương trình ứng dụng viết bằng ngôn ngữ Java chỉ cần chạy được trên máy ảo Java là có thể chạy được trên bất kỳ máy tính, hệ điều hành nào có máy ảo Java. "Viết một lần, chạy mọi nơi***(write once run anywhere).***
- **Hỗ trợ mạnh cho việc phát triển ứng dụng:** Công nghệ Java phát triển mạnh mẽ nhờ vào "đại gia Sun Microsystem" cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại hình ứng dụng khác nhau cụ thể như: J2SE (Java 2 Standard Edition) hỗ trợ phát triển những ứng dụng đơn, ứng dụng client-server; J2EE (Java 2 Enterprise Edition) hỗ trợ phát triển các ứng dụng thương mại, J2ME (Java 2 Micro Edition) hỗ trợ phát triển các ứng dụng trên các thiết bị di động, không dây, …

1.5. Công cụ soạn thảo mã nguồn Java.

Để viết mã nguồn java chúng ta có thể sử dụng trình soạn thảo NotePad hoặc một số môi trường phát triển hỗ trợ ngôn ngữ java như: Jbuilder của hãng Borland, Visual Café của hãng Symantec, JDeveloper của hãng Oracle, Visual J++ của Microsoft, NetBean.....

1.6. Các ứng dụng trong Java.

1.6.1 Java và ứng dụng Console

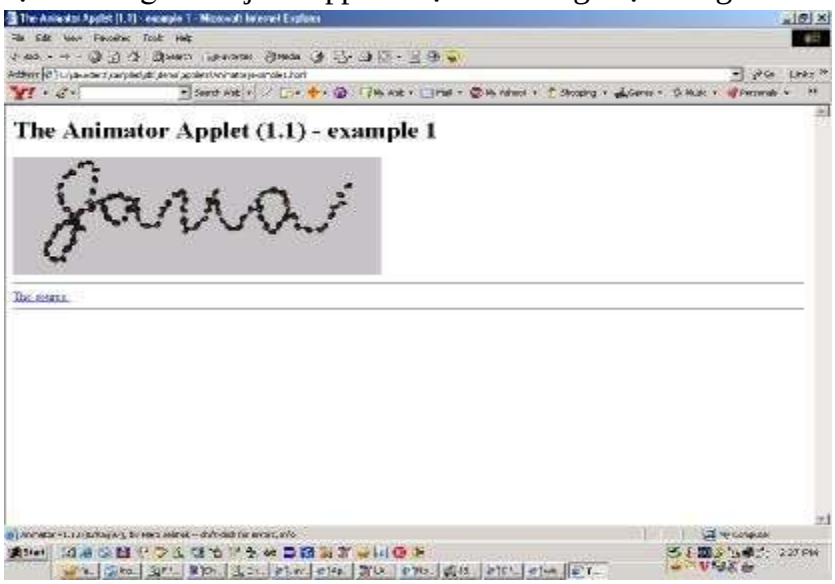
- Ứng dụng Console là ứng dụng nhập xuất ở chế độ văn bản tương tự như màn hình Console của hệ điều hành MS-DOS. Loại chương trình ứng dụng này thích hợp với những ai bước đầu làm quen với ngôn ngữ lập trình java.

- Các ứng dụng kiểu Console thường được dùng để minh họa các ví dụ cơ bản liên quan đến cú pháp ngôn ngữ, các thuật toán, và các chương trình ứng dụng không cần thiết đến giao diện người dùng đồ họa.

```
class HelloWorld
{       public static void main(String[] args)
        {
                System.out.println("\nHello World");
        }
}
```

1.6.2 Java và ứng dụng Applet

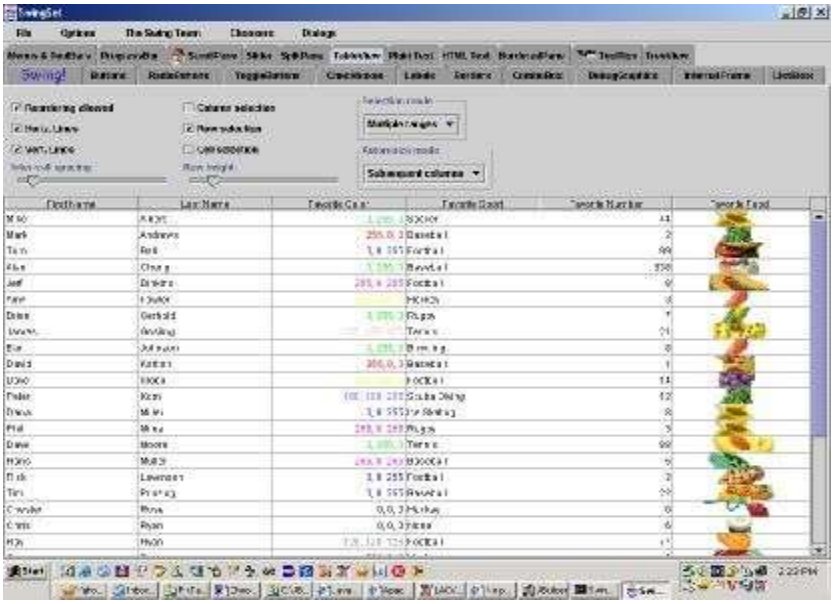
- Java Applet là loại ứng dụng có thể nhúng và chạy trong trang web của một trình duyệt web. Từ khi internet mới ra đời, Java Applet cung cấp một khả năng lập trình mạnh mẽ cho các trang web. Nhưng gần đây khi các chương trình duyệt web đã phát triển với khả năng lập trình bằng VB Script, Java Script, HTML, DHTML, XML, … cùng với sự canh tranh khốc liệt của Microsoft và Sun đã làm cho Java Applet lu mờ. Và cho đến bây giờ gần như các lập trình viên đều không còn "mặn mà" với Java Applet nữa. (trình duyệt IE đi kèm trong phiên bản Windows 2000 đã không còn mặc nhiên hỗ trợ thực thi một ứng dụng Java Applet). Hình bên dưới minh họa một chương trình java applet thực thi trong một trang web.



1.6.3. Java và phát triển ứng dụng Desktop dùng AWT và JFC

- Việc phát triển các chương trình ứng dụng có giao diện người dùng đồ họa trực quan giống như những chương trình được viết dùng ngôn ngữ lập trình VC++ hay Visual Basic đã được java giải quyết bằng thư viện AWT và JFC. JFC là thư viện rất phong phú và hỗ trợ mạnh mẽ hơn nhiều so với AWT. JFC giúp cho người lập trình có thể tạo ra một giao diện trực quan của bất kỳ ứng dụng nào. Liên quan đến việc phát triển các ứng dụng có giao diện người dùng đồ họa trực quan.

- Minh họa thiết kế giao diện người dùng sử dụng JFC



1.6.4. Java và phát triển ứng dụng Web

Java hỗ trợ mạnh mẽ đối với việc phát triển các ứng dụng Web thông qua công nghệ J2EE (Java 2 Enterprise Edition). Công nghệ J2EE hoàn toàn có thể tạo ra các ứng dụng Web một cách hiệu quả không thua kém công nghệ .NET mà Microsft đang quảng cáo.

1.6.5. Java và phát triển các ứng dụng nhúng

Java Sun đưa ra công nghệ J2ME (The Java 2 Platform, Micro Edition J2ME) hỗ trợ phát triển các chương trình, phần mềm nhúng. J2ME cung cấp một môi trường cho những chương trình ứng dụng có thể chạy được trên các thiết bị cá nhân như: điện thoại di động, máy tính bỏ túi PDA hay Palm, cũng như các thiết bị nhúng khác.

CHƯƠNG II

CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH JAVA

2.1.Biến

• *Biến là*: vùng nhớ dùng để lưu trữ các giá trị của chương trình. Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến.

- Tên biến thông thường là một chuỗi các ký tự (Unicode), ký số.
- Tên biến phải bắt đầu bằng một chữ cái, một dấu gạch dưới hay dấu dollar.
- Tên biến không được trùng với các từ khóa (xem phụ lục các từ khóa trong java).
- Tên biến không có khoảng trắng ở giữa tên.

Trong java, biến có thể được khai báo ở bất kỳ nơi đâu trong chương trình.

+ Cách khai báo:

<kiểu_dữ_liệu> <tên_biến>;

<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;

Gán giá trị cho biến

<tên_biến> = <giá_trị>;

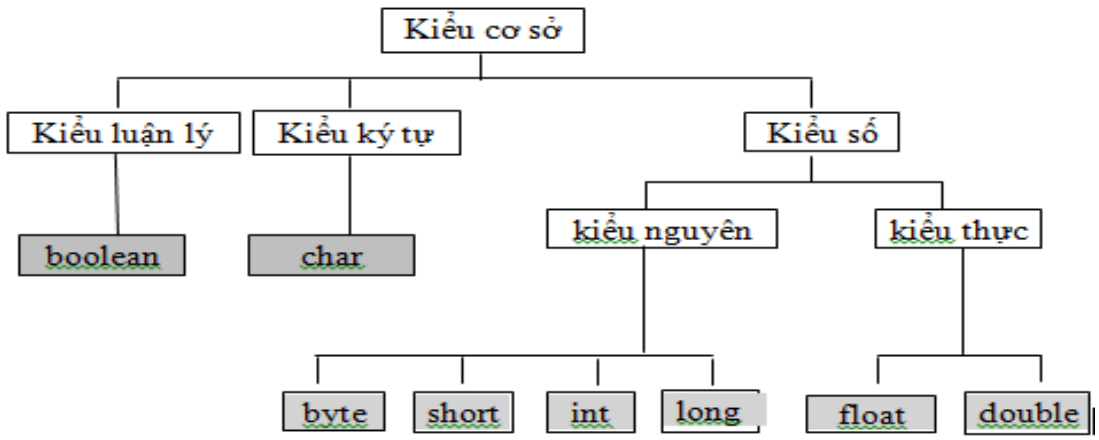
• *Biến công cộng (toàn cục):* là biến có thể truy xuất ở khắp nơi trong chương trình, thường được khai báo dùng từ khóa public, hoặc đặt chúng trong một class.

• *Biến cục bộ*: là biến chỉ có thể truy xuất trong khối lệnh nó khai báo.

Lưu ý: Trong ngôn ngữ lập trình java có phân biệt chữ in hoa và in thường. Vì vậy chúng ta cần lưu ý khi đặt tên cho các đối tương dữ liệu cũng như các xử lý trong chương trình**.**

2.2. Các kiểu dữ liệu cơ sở

- Ngôn ngữ lập trình java có 8 kiểu dữ liệu cơ sở: *byte, short, int, long, float, double, boolean và char*.



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

Một số lưu ý đối với các phép toán trên số nguyên:

- Nếu hai toán hạng kiểu long thì kết quả là kiểu long. Một trong hai toán hạng không phải kiểu long sẽ được chuyển thành kiểu long trước khi thực hiện phép toán.
- Nếu hai toán hạng đầu không phải kiểu long thì phép tính sẽ thực hiện với kiểu int.
- Các toán hạng kiểu byte hay short sẽ được chuyển sang kiểu int trước khi thực hiện phép toán.
- Trong java không thể chuyển biến kiểu int và kiểu boolean như trong ngôn ngữ C/C++.

***Ví dụ*: có đoạn chương trình như sau**

```
boolean b = false;
if (b == 0)
{
        System.out.println("Xin chao");
}
```

Lúc biên dịch đoạn chương trình trên trình dịch sẽ báo lỗi: không được phép so sánh biến kiểu boolean với một giá trị kiểu int.

2.2.1. Kiểu dấu chấm động

- Đối với kiểu dấu chấm động hay kiểu thực, java hỗ trợ hai kiểu dữ liệu là float và double.
- Kiểu float có kích thước 4 byte và giá trị mặc định là 0.0f Kiểu double có kích thước 8 byte và giá trị mặc định là 0.0d
- Số kiểu dấu chấm động không có giá trị nhỏ nhất cũng không có giá trị lớn nhất. Chúng có thể nhận các giá trị:
  + Số âm
  + Số dương
  +Vô cực âm
  +Vô cực dương
- Khai báo và khởi tạo giá trị cho các biến kiểu dấu chấm động:

+ float x = 100.0/7;

+ double y = 1.56E6;

- **Một số lưu ý đối với các phép toán trên số dấu chấm động:**

- Nếu mỗi toán hạng đều có kiểu dấn chấm động thì phép toán chuyển thành phép toán dấu chấm động.

- Nếu có một toán hạng là double thì các toán hạng còn lại sẽ được chuyển thành kiểu double trước khi thực hiện phép toán.

- Biến kiểu float và double có thể ép chuyển sang kiểu dữ liệu khác trừ kiểu boolean.

2.2.2. Kiểu ký tự (char)

- Kiểu ký tự trong ngôn ngữ lập trình java có kích thước là 2 bytes và chỉ dùng để biểu diễn các ký tự trong bộ mã Unicode. Như vậy kiểu char trong java có thể biểu diễn tất cả 2^{16} = 65536 ký tự khác nhau.

- Giá trị mặc định cho một biến kiểu char là null.

2.2.3. Kiểu luận lý (boolean)

- Kiểu boolean chỉ nhận 1 trong 2 giá trị: true hoặc false.

- Trong java kiểu boolean không thể chuyển thành kiểu nguyên và ngược lại.

- Giá trị mặc định của kiểu boolean là false.

2.3.Hằng

- Hằng là một giá trị bất biến trong chương trình

- Tên hằng được đặt theo qui ước giống như tên biến.

- Hằng số nguyên: trường hợp giá trị hằng ở dạng long ta thêm vào cuối chuỗi số chữ "l" hay "L". (ví dụ: 1L)

- Hằng số thực: trường hợp giá trị hằng có kiểu float ta thêm tiếp vĩ ngữ "f" hay "F", còn kiểu số double thì ta thêm tiếp vĩ ngữ "d" hay "D".

- Hằng Boolean: java có 2 hằng boolean là true, false.

- Hằng ký tự: là một ký tự đơn nằm giữa nằm giữa 2 dấu ngoặc đơn.

+Ví dụ: 'a': hằng ký tự a

+Một số hằng ký tự đặc biệt

Ký tự	Ý nghĩa
\b	Xóa lùi (BackSpace)
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\"	Nháy kép
\'	Nháy đơn
\\	Số ngược
\f	Đẩy trang
\uxxxx	Ký tự unicode

- **Hằng chuỗi:** là tập hợp các ký tự được đặt giữa hai dấu nháy kép “”. Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng.

+ Ví dụ: “Hello Wolrd”

- Lưu ý: Hằng chuỗi không phải là một kiểu dữ liệu cơ sở nhưng vẫn được khai báo và sử dụng trong các chương trình.

2.4. Lệnh, khối lệnh trong java

- Giống như trong ngôn ngữ C, các câu lệnh trong java kết thúc bằng một dấu chấm phẩy (;).
- Một khối lệnh là đoạn chương trình gồm hai lệnh trở lên và được bắt đầu bằng dấu mở ngoặc nhọn ({) và kết thúc bằng dấu đóng ngoặc nhọc (}).
- Bên trong một khối lệnh có thể chứa một hay nhiều lệnh hoặc chứa các khối lệnh khác.

```
{ // khối 1
        { // khối 2
            lệnh 2.1
            lệnh 2.2
            ...
        } // kết thúc khối lệnh 2
        lệnh 1.1
        lệnh 1.2
        ...
} // kết thúc khối lệnh 1

{ // bắt đầu khối lệnh 3
        // Các lệnh thuộc khối lệnh 3
        // ...
} // kết thúc thối lệnh 3
```

2.5 Toán tử và biểu thức

2.5.1. Toán tử số học

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

2.5.2. Toán tử trên bit

Toán tử	Ý
&	AND
|	OR
^	XOR
<<	Dịch trái
>>	Dịch phải
>>>	Dịch phải và điền 0 vào bit trống
~	Bù bit

2.5.3. Toán tử quan hệ & logic

Toán tử	Ý nghĩa
==	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng
||	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

2.5.4. Toán tử ép kiểu

- Ép kiểu rộng (widening conversion): từ kiểu nhỏ sang kiểu lớn (không mất mát thông tin)
- Ép kiểu hẹp (narrow conversion): từ kiểu lớn sang kiểu nhỏ (có khả năng mất mát thông tin)

<tên biến> = (kiểu_dữ_liệu) <tên_biến>;

Ví dụ:

Float fNum = 2.2;

2.5.5. Toán tử điều kiện

Cú pháp:

<điều kiện> ? <biểu thức 1> : < biểu thức 2>

- Nếu điều kiện đúng thì có giá trị, hay thực hiện <biểu thức 1>, còn ngược lại là <biểu thức 2>.
- <điều kiện>: là một biểu thức logic
- <biểu thức 1>, <biểu thức 2>: có thể là hai giá trị, hai biểu thức hoặc hai hành động.

Ví dụ:

```
int x = 10; int y =
20;
int Z = (x<y) ? 30 : 40;
// Kết quả z = 30 do biểu thức (x < y) là đúng
```

2.6. Cấu trúc điều khiển

2.6.1. Cấu trúc điều kiện if … else

❖ **Dạng 1:**

```
if (<điều_kiện>)
{
    <khối_lệnh>;
}
```

❖ **Dạng 2:**

```
if (<điều_kiện>)
{
 <khối _lệnh1>;
 }
else
 {
    <khối _lệnh2>;
 }
```

2.6.2. Cấu trúc switch … case

```
switch (<biến>)
{
    case <giátrị_1>:<khối_lệnh_1>; break;
          …….
     case <giátrị_1>:<khối_lệnh_1>; break;

  default: <khối lệnh default>;
 }
```

2.6.3. Cấu trúc vòng lặp

❖ **Dạng 1**: while(…)

```
while (điều_kiện_lặp)
{
        khối _lệnh;
}
```

❖ **Dạng 2**: do { … } while;

```
do
{
        khối_lệnh;
} while (điều_kiện);
```

❖ **Dạng 3**: for (…)

```
for (khởi_tạo_biến_đếm;đk_lặp;tăng_biến)
{
    <khối _lệnh>;
}
```

2.6.4 Cấu trúc lệnh nhảy (jum)

- **Lệnh break:** trong cấu trúc switch chúng ta dùng câu lệnh break để thoát thỏi cấu trúc switch trong cùng chứa nó. Tương tự như vậy, trong cấu trúc lặp, câu lệnh break dùng để thóat khỏi cấu trúc lặp trong cùng chứa nó.
- **Lệnh continue:** dùng để tiếp tục vòng lặp trong cùng chứa nó (ngược với break).
- **Nhãn (label):** Không giống như C/C++, Java không hỗ trợ lệnh goto để nhảy đến 1 vị trí nào đó của chương trình. Java dùng kết hợp nhãn ***(label)*** với từ khóa ***break*** và ***continue*** để thay thế cho lệnh goto.

Ví dụ:

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

Lệnh "*label*": xác định vị trí của nhãn và xem như tên của vòng lặp ngoài. Nếu *<biểu thức điều kiện>* đúng thì lệnh *break label* sẽ thực hiện việc nhảy ra khỏi vòng lặp có nhãn là "*label*", ngược lại sẽ tiếp tục vòng lặp có nhãn "*label*" (khác với *break* và *continue* thông thường chỉ thoát khỏi hay tiếp tục vòng lặp trong cùng chứa nó.).

2.7. Kiểu dữ liệu mảng

- Như chúng ta đã biết Java có 2 kiểu dữ liệu:
  - Kiểu dữ liệu cơ sở (Primitive data type)
  - Kiểu dữ liệu tham chiếu hay dẫn xuất (reference data type): thường có 3 kiểu:
    - Kiểu mảng
    - Kiểu lớp
    - Kiểu giao tiếp(interface).
- Ở đây chúng ta sẽ tìm hiểu một số vấn đề cơ bản liên quan đền kiểu mảng. Kiểu lớp(class) và giao tiếp(interface) chúng ta sẽ tìm hiểu chi tiết trong chương 3 và các chương sau.

2.7.1. Khái niệm mảng

- Mảng là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số của nó trong mảng.

2.7.2. Khai báo mảng

<kiểu dữ liệu> <tên mảng>[];

hoặc <kiểu dữ liệu>[] <tên mảng>

Ví dụ : int arrInt[];

hoặc int[] arrInt;

int[] arrInt1, arrInt2, arrInt3;

2.7.3. Cấp phát bộ nhớ cho mảng

❖ Không giống như trong C, C++ kích thước của mảng được xác định khi khai báo. Chẳng hạn như:

int arrInt[100]; // Khai báo náy trong Java sẽ bị báo lỗi.

- Để cấp phát bộ nhớ cho mảng trong Java ta cần dùng từ khóa new. (Tất cả trong Java đều thông qua các đối tượng). Chẳng hạn để cấp phát vùng nhớ cho mảng trong Java ta làm như sau:

*int arrInt = **new** int[100];*

2.7.4. Khởi tạo mảng

Chúng ta có thể khởi tạo giá trị ban đầu cho các phần tử của mảng khi nó được khai báo**.**

Ví dụ :

```
int    arrInt[]    = {1, 2, 3}; char
       arrChar[]   = {'a', 'b', 'c'};
String arrStrng[]  = {"ABC", "EFG", 'GHI'};
```

2.7.5. Truy cập mảng

Chỉ số mảng trong Java bắt đầu tư 0. Vì vậy phần tử đầu tiên có chỉ số là 0, và phần tử thứ n có chỉ số là n-1. Các phần tử của mảng được truy xuất thông qua chỉ số của nó đặt giữa cặp dấu ngoặc vuông ([]).

Ví dụ*:*

```
int arrInt[] = {1, 2, 3};
int x = arrInt[0]; // x sẽ có giá trị là 1.
int y = arrInt[1]; // y sẽ có giá trị là 2.
int z = arrInt[2]; // z sẽ có giá trị là 3.
```

Lưu ý: Trong nhưng ngôn ngữ lập trình khác (C chẳng hạn), một chuỗi được xem như một mảng các ký tự. Trong java thì khác, java cung cấp một lớp String để làm việc với đối tượng dữ liệu chuỗi cùng khác thao tác trên đối tượng dữ liệu này.

*2.7.6. Một số ví dụ***:**

❖ **Ví dụ 1 :***Nhập và xuất giá trị các phần tử của một mảng các số nguyên :*

class ArrayDemo

```
{
    public static void main(String args[])
    {
        int arrInt[] = new int[10];
        int i;
        for(i = 0; i < 10; i = i+1)
            arrInt[i] = i;
        for(i = 0; i < 10; i = i+1)
            System.out.println("This is arrInt[" + i
            + "]: " + arrInt[i]);
    }
}
```

❖ ***Ví dụ 2***: Nhập ký tự từ bàn phím

```
import java.io.*;
/* gói này cung cấp thự viện xuất nhập hệ thống thông qua
những luồng dữ //liệu và hệ thống file.*/
class InputChar
{
    public static void main(String args[])
    {
        char ch = '';
        try
        {
            ch = (char) System.in.read();
        }
        catch(Exception e)
        {
            System.out.println("Nhập lỗi!");
        }

        System.out.println("Ky tu vua nhap:" + ch);
    }
}
```

CHƯƠNG III
HƯỚNG ĐỐI TƯỢNG TRONG JAVA

3.1. Mở đầu

OOP là một trong những tiếp cận mạnh mẽ, và rất hiệu quả để xây dựng nên những chương trình ứng dụng trên máy tính. Từ khi ra đời cho đến nay lập trình OOP đã chứng tỏ được sức mạnh, vai trò của nó trong các đề án tin học.

3.2. Lớp (Class)

3.2.1.Khái niệm

- Chúng ta có thể xem lớp như một khuôn mẫu (template) của đối tượng (Object). Trong đó bao gồm dữ liệu của đối tượng (fields hay properties) và các phương thức(methods) tác động lên thành phần dữ liệu đó gọi là các phương thức của lớp.
- Các đối tượng được xây dựng bởi các lớp nên được gọi là các thể hiện của lớp (class instance).

3.2.2. Khai báo/định nghĩa lớp

```
class <ClassName>
{
        <kiểu dữ liệu> <field_1>;
        <kiểu dữ liệu> <field_2>;
       constructor
        method_1
       method_2
}
```

- ***class*:** là từ khóa của java
- ***ClassName*:** là tên chúng ta đặt cho lớp
- ***field_1, field_2*:** các thuộc tính, các biến, hay các thành phần dữ liệu của lớp.
- ***constructor*:** là sự xây dựng, khởi tạo đối tượng lớp.
- ***method_1, method_2***: là các phương thức/hàm thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

3.2.3. Tạo đối tượng của lớp

ClassName objectName = **new** *ClassName()*;

3.2.4. Thuộc tính của lớp

- Vùng dữ liệu (fields) hay thuộc tính (properties) của lớp được khai báo bên trong lớp như sau:

```
class <ClassName>
{
// khai báo những thuộc tính của lớp
<tiền tố> <kiểu dữ liệu> field1;
// ...
```

}

- Để xác định quyền truy xuất của các đối tượng khác đối với vùng dữ liệu của lớp người ta thường dùng 3 tiền tố sau:

- **public:** có thể truy xuất từ tất cả các đối tượng khác
- **private:** một lớp không thể truy xuất vùng private của 1 lớp khác.
- **protected:** vùng protected của 1 lớp chỉ cho phép bản thân lớp đó và những lớp dẫn xuất từ lớp đó truy cập đến.

- Ví dụ:

```
public class xemay
{       Public          String      nhasx;
        public          String      model;
        private         float       chiphisx;
        Protected       int         thoigiansx;
}
```

- Lưu ý: Thông thường để an toàn cho vùng dữ liệu của các đối tượng người ta tránh dùng tiền tố public, mà thường chọn tiền tố private để ngăn cản quyền truy cập đến vùng dữ liệu của một lớp từ các phương thức bên ngoài lớp đó.

Hàm – phương thức lớp (method)

Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

3.2.5. Hàm - Phương thức lớp (Method)

Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

❖ ***Khai báo phương thức:***

```
<Tiền tố> <kiểu trả về> <Tên phương thức> (<danh sách đối số>)
{
        <khối lệnh>;
}
```

Để xác định quyền truy xuất của các đối tượng khác đối với các phương thức của lớp người ta thường dùng các tiền tố sau:

- **public:** phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- **protected:** có thể truy cập được từ lớp khai báo và những lớp dẫn xuất từ nó.
- **private:** chỉ được truy cập bên trong bản thân lớp khai báo.
- **static:** phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có nghĩa là phương thức đó có thể được thực hiện kể cả khi không có đối tượng của lớp chứa phương thức đó.

- **final:** phương thức có tiền tố này không được khai báo chồng ở các lớp dẫn xuất.
- **abstract:** phương thức không cần cài đặt (không có phần source code), sẽ được hiện thực trong các lớp dẫn xuất từ lớp này.
- **synchoronized:** dùng để ngăn các tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình miltithreads.

<kiểu trả về>: có thể là kiểu void, kiểu cơ sở hay một lớp.

<Tên phương thức>: đặt theo qui ước giống tên biến.

<danh sách thông số>: có thể rỗng

❖ ***Lưu ý:***

- Thông thường trong một lớp các phương thức nên được khai báo dùng từ khóa public, khác với vùng dữ liệu thường là dùng tiền tố private vì mục đích an toàn.

- Những biến nằm trong một phương thức của lớp là các biến cục bộ (local) và nên được khởia tạo sau khi khai báo.

❖ ***Ví dụ:***

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

3.2.6. Khởi tạo một đối tượng (constructor)

- Contructor thật ra là một loại phương thức đặc biệt của lớp. Constructor dùng gọi tự động khi khởi tạo một thể hiện của lớp, có thể dùng để khởi gán những giá trị măc định. Các constructor không có giá trị trả về, và có thể có tham số hoặc không có tham số.

- Constructor phải có cùng tên với lớp và được gọi đến dùng từ khóa new.
- Nếu một lớp không có constructor thì java sẽ cung cấp cho lớp một constructor mặc định (default constructor). Những thuộc tính, biến của lớp sẽ được khởi tạo bởi các giá trị mặc định (số: thường là giá trị 0, kiểu luận lý là giá trị false, kiểu đối tượng giá trị null, …)

Lưu ý: thông thường để an toàn, dễ kiểm soát và làm chủ mã nguồn chương trình chúng ta nên khai báo một constructor cho lớp.

Ví dụ:

```
public class xemay
{
// ...
public xemay(){
}
public xemay(String s_nhasx, String s_model, f_chiphisx, int i_thoigiansx,
                int i_so);
{
    nhasx = s_nhasx; model = s_model;
    chiphisx = f_chiphisx;
    thoigiansx = i_thoigiansx; so = i_so;
    // hoặc
    // this.nhasx = s_nhasx;
    // this.model = s_model;
    // this.chiphisx = f_chiphisx;
    // this.thoigiansx = i_thoigiansx;
    // this.so = i_so;
}
}
```

3.2.7. Biến this

Biến this là một biến ẩn tồn tại trong tất cả các lớp trong ngông ngữ java. Một class trong Java luôn tồn tại một biến this, biến this được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

3.2.8 Khai báo chồng phương thức (overloading method)

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

Ví dụ:

```
public class xemay
{       // khai báo fields ...
        public float tinhgiaban()
        {       return 2 * chiphisx;
```

```
        }
        public float tinhgiaban(float huehong)
        {       return (2 * chiphisx + huehong);
        }
}
```

3.3. Đặc điểm hướng đối tượng của java

- Hỗ trợ những nguyên tắc cơ bản của lập trình hướng đối tượng, tất cả các ngôn ngữ lập trình kể cả java đều có ba đặc điểm chung: tính đóng gói (encapsulation), tính đa hình (polymorphism), và tính kế thừa (inheritance).

3.3.1 Đóng gói (encapsulation)

- Cơ chế đóng gói trong lập trình hướng đối tượng giúp cho các đối tượng dấu đi một phần các chi tiết cài đặt, cũng như phần dữ liệu cục bộ của nó, và chỉ công bố ra ngoài những gì cần công bố để trao đổi với các đối tượng khác. Hay chúng ta có thể nói đối tượng là một thành tố hỗ trợ tính đóng gói.

- Đơn vị đóng gói cơ bản của ngôn ngữ java là class. Một class định nghĩa hình thức của một đối tượng. Một class định rõ những thành phần dữ liệu và các đoạn mã cài đặt các thao tác xử lý trên các đối tượng dữ liệu đó. Java dùng class để xây dựng những đối tượng. Những đối tượng là những thể hiện (instances) của một class.

- Một lớp bao gồm thành phần dữ liệu và thành phần xử lý. Thành phần dữ liệu của một lớp thường bao gồm các biến thành viên và các biến thể hiện của lớp. Thành phần xử lý là các thao tác trên các thành phần dữ liệu, thường trong java người gọi là phương thức. Phương thức là một thuật ngữ hướng đối tượng trong java, trong C/C++ người ta thường dùng thuật ngữ là hàm.

3.3.2. Tính đa hình(polymorphism):

Tính đa hình cho phép cài đặt các lớp dẫn xuất khác nhau từ một lớp nguồn. Một đối tượng có thể có nhiều kiểu khác nhau gọi là tính đa hình.

3.3.3. Tính kế thừa

- Một lớp con (subclass) có thể kế thừa tất cả những vùng dữ liệu và phương thức của một lớp khác (siêu lớp - superclass). Như vậy việc tạo một lớp mới từ một lớp đã biết sao cho các thành phần (fields và methods) của lớp cũ cũng sẽ thành các thành phần (fields và methods) của lớp mới. Khi đó ta gọi lớp mới là lớp dẫn xuất (derived class) từ lớp cũ (superclass). Có thể lớp cũ cũng là lớp được dẫn xuất từ một lớp nào đấy, nhưng đối với lớp mới vừa tạo thì lớp cũ đó là một lớp siêu lớp trực tiếp (immediate supperclass).

- Dùng từ khóa extends để chỉ lớp dẫn xuất.

```
class A extends B
{
// ...
```

```
}
```

3.3.3.1. Khai báo phương thức chồng

Tính kế thừa giúp cho các lớp con nhận được các thuộc tính/phương thức public và protected của lớp cha. Đồng thời cũng có thể thay thế các phương thức của lớp cha bằng cách khai báo chồng. Chẳng hạn phương thức *tinhgiaban()* áp dụng trong lớp *xega* sẽ cho kết quả gấp 2.5 lần chi phí sản xuất thay vì gấp 2 chi phí sản xuất giống như trong lớp *xemay*.

***Ví dụ**:*

```
public class xega extends xemay
{
        public xega()
        {
        }
        public xega(String s_nhasx, String s_model,
        f_chiphisx, int i_thoigiansx);
        {
                this.nhasx = s_nhasx;
                this.model = s_model;
                this.chiphisx = f_chiphisx;
                this.thoigiansx = i_thoigiansx;
                this.so = 0;
        }

        public float tinhgiaban()
        {
                return 2.5 * chiphisx;
        }
}
```

Java cung cấp 3 tiền tố/từ khóa để hỗ trợ tính kế thừa của lớp:

- **public:** lớp có thể truy cập từ các gói, chương trình khác.
- **final:** Lớp hằng, lớp không thể tạo dẫn xuất (không thể có con), hay đôi khi người ta gọi là lớp "vô sinh".
- **abstract:** Lớp trừu tượng (không có khai báo các thành phần và các phương thức trong lớp trừu tượng). Lớp dẫn xuất sẽ khai báo, cài đặt cụ thể các thuộc tính, phương thức của lớp trừu tượng.

3.3.3.2. Lớp nội

Lớp nội là lớp được khai báo bên trong 1 lớp khác. Lớp nội thể hiện tính đóng gói cao và có thể truy xuất trực tiếp biến của lớp cha.

***Ví dụ**:*

```
public class A
{
        // ...
        int <field_1>
        static class B
        {
                // ...
                int <field_2>
                public B(int par_1)
                {
                        field_2 = par_1 + field_1;
                }
        }
}
```

Trong ví dụ trên thì chương trình dịch sẽ tạo ra hai lớp với hai files khác nhau: A.class và B.class

3.3.3.3. Lớp vô sinh

- Lớp không thể có lớp dẫn xuất từ nó (không có lớp con) gọi là lớp "vô sinh", hay nói cách khác không thể kế thừa được từ một lớp "vô sinh". Lớp "vô sinh" dùng để hạn chế, ngăn ngừa các lớp khác dẫn xuất từ nó.

- Để khai báo một lớp là lớp "vô sinh", chúng ta dùng từ khóa final class.

- Tất cả các phương thức của lớp vô sinh đều vô sinh, nhưng các thuộc tính của lớp vô sinh thì có thể không vô sinh.

Ví dụ:

```
public final class A
{
        public final int x;
        private int y;
        public final void method_1()
        {
                // ...
        }
        public final void method_2()
        {
```

```
                    // ...
                }
            }
```

3.3.3.4. Lớp trừu tượng

Lớp trừu tượng là lớp không có khai báo các thuộc tính thành phần và các phương thức. Các lớp dẫn xuất của nó sẽ khai báo thuộc tính, cài đặt cụ thể các phương thức của lớp trừu tượng.

Ví dụ:

```
        abstract class A
        {
abstract void method_1();
        }
        public class B extends A
        {
            public void method_1()
            {
                // cài đặt chi tiết cho phương thức method_1
                // trong lớp con B.
                // ...
            }
        }
        public class C extends A
        {
            public void method_1()
            {
                // cài đặt chi tiết cho phương thức method_1
                // trong lớp con C.
                // ...
            }
        }
```

➢ **Lưu ý:** Các phương thức được khai báo dùng các tiền tố private và static thì không được khai báo là trừu tượng abstract. Tiền tố private thì không thể truy xuất từ các lớp dẫn xuất, còn tiền tố static thì chỉ dùng riêng cho lớp khai báo mà thôi.

3.3.3.5. Phương thức finalize

- Trong java không có kiểu dữ liệu con trỏ như trong C, người lập trình không cần phải quá bận tâm về việc cấp phát và giải phóng vùng nhớ, sẽ có một trình dọn dẹp hệ thống đảm trách việc này. Trình dọn dẹp hệ thống sẽ dọn dẹp vùng nhớ cấp phát cho các đối tượng trước khi hủy một đối tượng.

- Phương thức *finalize()* là một phương thức đặc biệt được cài đặt sẵn cho các lớp. Trình dọn dẹp hệ thống sẽ gọi phương thức này trước khi hủy một đối tượng. Vì vậy việc cài đặt một số thao tác giải phóng, dọn dẹp vùng nhớ đã cấp phát cho các đối tượng dữ liệu trong phương thức *finalize()* sẽ giúp cho người lập trình chủ động kiểm soát tốt quá trình hủy đối tượng thay vị giao cho trình dọn dẹp hệ thống tự động. Đồng thời việc cài đặt trong phương thức *finalize()* sẽ giúp cho bộ nhớ được giải phóng tốt hơn, góp phần cải tiến tốc độ chương trình.

3.4. Gói (packages)

- Việc đóng gói các lớp lại tạo thành một thư viện dùng chung gọi là package.

- Một package có thể chứa một hay nhiều lớp bên trong, đồng thời cũng có thể chứa một package khác bên trong.

- Để khai báo một lớp thuộc một gói nào đấy ta phải dùng từ khóa package.

- Dòng khai báo gói phải là dòng đầu tiên trong tập tin khai báo lớp.

- Các tập tin khai báo lớp trong cùng một gói phải được lưu trong cùng một thư mục.

Lưu ý: Việc khai báo import tất cả các lớp trong gói sẽ làm tốn bộ nhớ. Thông thường chúng ta chỉ nên import những lớp cần dùng trong chương trình.

3.5. Giao diện (interface)

3.5.1. Khái niệm interface:

- Như chúng ta đã biết một lớp trong java chỉ có một siêu lớp trực tiếp hay một cha duy nhất (đơn thừa kế). Để tránh đi tính phức tạp của đa thừa kế (multi-inheritance) trong lập trình hướng đối tượng, Java thay thế bằng giao tiếp (interface). Một lớp có thể có nhiều giao tiếp (interface) với các lớp khác để thừa hưởng thêm vùng dữ liệu và phương thức của các giao tiếp này**.**

3.5.2. Khai báo interface:

- Interface được khai báo như một lớp. Nhưng các thuộc tính của interface là các hằng (khai báo dùng từ khóa final) và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

- Trong các lớp có cài đặt các interface ta phải tiến hành cài đặt cụ thể các phương thức này.

Ví dụ*:*

```
public interface sanpham
{       static final String nhasx = "Honda VN";
        static fina   String dienthoai = "08-
        publi  int    8123456"; gia(String s, model);
}
// khai báo 1 lớp có cài đặt interface
public class xemay implements
```

```
sanpham
{       // cài đặt lại phương thức của giao diện trong
        lớp  public  int gia(String s_model)
        {
                if (s_model.equals("2005"))
                        return (2000);
                else
                        return (1500);
        }

        public  String chobietnhasx()
        {
                return (nhasx);
        }
}
```

- Có một vấn đề khác với lớp là một giao diện (interface) không chỉ có một giao diện cha trực tiếp mà có thể dẫn xuất cùng lúc nhiều giao diện khác (hay có nhiều giao diện cha). Khi đó nó sẽ kế thừa tất cả các giá trị hằng và các phương thức của các giao diện cha. Các giao diện cha được liệt kê thành chuỗi và cách nhau bởi dấu phẩy ",". Khai báo như sau:

```
public interface InterfaceName extends interface1,
interface2,  interface3
{
        // ...
}
```

CHƯƠNG IV

THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

4.1 Giới thiệu thư viện awt

- Thư viện awt là bộ thư viện dùng để xây dựng giao diện người dùng cho một chương trình ứng dụng có đầy đủ các thành phần cơ bản như: Label, Button, Checkbox, Radiobutton, Choice, List, Text Field, Text Area, Scrollbar, Menu, Frame…

- Giống như các API của Windows, java cung cấp cho người lập trình thư viện awt. Nhưng khác với các hàm API, thư viện awt không phụ thuộc hệ điều hành. Thư viện awt là nền tảng, cơ sở giúp cho chúng ta tiếp cận với thư viện mở rộng JFC hiệu quả hơn.

- Cấu trúc cây phân cấp của tất cả những lớp trong thư viện awt chúng ta có thể xem chi tiết trong tài liệu kèm theo bộ công cụ j2se (phần API Specification)

4.2. Các khái niệm cơ bản

4.2.1.Component

Component là một đối tượng có biểu diễn đồ họa được hiển thị trên màn hình mà người dùng có thể tương tác được. Chẳng hạn như những nút nhấn (button), những checkbox, những scrollbar,… Lớp **Component** là một lớp trừu tượng.

java.lang.Object

java.awt.Component

4.2.2.Container

- Container: là đối tượng vật chứa hay những đối tượng có khả năng quản lý và nhóm các đối tượng khác lại. Những đối tượng con thuộc thành phần awt như: button, checkbox, radio button, scrollbar, list,… chỉ sử dụng được khi ta đưa nó vào khung chứa (container).

- Một số đối tượng container trong Java:

  - **Panel:** Đối tượng khung chứa đơn giản nhất, dùng để nhóm các đối tượng, thành phần con lại. Một Panel có thể chứa bên trong một Panel khác.

    java.lang.Object
    +--java.awt.Component
    +--java.awt.Container
    +--***java.awt.Panel***

  - **Frame:** khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.

    java.lang.Object
    +--java.awt.Component

+--java.awt.Container
+--java.awt.Window
+--**java.awt.Frame**

- **Dialogs:** đây là một cửa sổ dạng hộp hội thoại (cửa sổ dạng này còn được gọi là pop-up window), cửa sổ dạng này thường được dùng để đưa ra thông báo, hay dùng để lấy dữ liệu nhập từ ngoài vào thông qua các đối tượng, thành phần trên dialog như TextField chẳng hạn. Dialog cũng là một cửa sổ nhưng không đầy đủ chức năng như đối tượng khung chứa Frame.

java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--java.awt.Window
+--**java.awt.Dialog**

- **ScrollPanes**: là một khung chứa tương tự khung chứa Panel, nhưng có thêm 2 thanh trượt giúp ta tổ chức và xem được các đối tượng lớn choán nhiều chỗ trên màn hình như những hình ảnh hay văn bản nhiều dòng.

java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--**java.awt.ScrollPane**

4.2.3. Layout Manager

Khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗ ở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Các bộ quản lý trình bày mà thư viện AWT cung cấp cho ta bao gồm:

- **FlowLayout**: Sắp xếp các đối tượng *từ trái qua phải và từ trên xuống dưới*. Các đối tượng đều giữ nguyên kích thước của mình.
- **BorderLayout:** Các đối tượng được đặt theo các đường viền của khung chứa theo các cạnh *West, East, South, North* và *Center* tức Đông, Tây, Nam, Bắc và Trung tâm hay Trái, Phải, Trên, Dưới và Giữa tùy theo cách nhìn của chúng ta**.**
- **GridLayout:** Tạo một khung lưới vô hình với các ô bằng nhau. Các đối tượng sẽ đặt vừa kích thước với từng ô đó. Thứ tự sắp xếp cũng từ trái qua phải và từ trên xuống dưới.
- **GridBagLayout:** Tương tự như GridLayout, các đối tượng khung chứa cũng được đưa vào một lưới vô hình. Tuy nhiên kích thước các đối tượng không

nhất thiết phải vừa với 1 ô mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉ định thông qua đối tượng GridBagConstraint.

• **Null Layout:** Cách trình bày tự do. Đối với cách trình bày này người lập trình phải tự động làm tất cả từ việc định kích thước của các đối tượng, cũng như xác định vị trí của nó trên màn hình. Ta không phụ thuộc vào những ràng buộc đông, tây , nam, bắc gì cả.

4.3. Thiết kế GUI cho chương trình

4.3.1.Tạo khung chứa cửa sổ chương trình

Thông thường để tạo cửa sổ chính cho chương trình ứng dụng ta tiến hành các bước:

- Tạo đối tượng Frame
- Xác định kích thước của Frame
- Thể hiện Frame trên màn hình

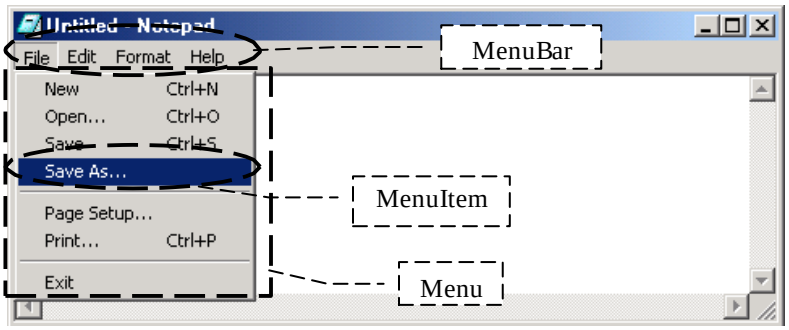
Ví dụ:

```
import java.awt.*;
class FrameDemo
{
    public static void main(String args[])
    {
            // Tạo đối tượng khung chứaFrame
            Frame fr = new Frame("My First Window") ;
            // Xác định kích thước, vị trí của Frame
            fr.setBounds(0, 0, 640, 480);
            // Hiển thị Frame
            fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:

4.3.2. Tạo hệ thống thực đơn

Đối với thư viện awt, để xây dựng hệ thống thực đơn cho chương trình ứng dụng chúng ta có thể dùng các lớp MenuBar, Menu, MenuItem, MenuShortcut.



Ví dụ**: Tạo hệ thống thực đơn cho chương trình Calculator**

```
import java.awt.*;
import java.awt.event.*;
class Calculator
{
    public static void main(String[] args)
    {
        createMenu();
    }
    private static void createMenu()
    {
        // Tao Frame ung dung
        final Frame fr = new Frame();
        fr.setLayout(new BorderLayout());

        // Tao cac menu bar
        MenuBar menu = new MenuBar();
        Menu menuFile = new Menu("Edit");
        MenuItem copyItem = new MenuItem("Copy
        Ctrl+C"); MenuItem pasteItem = new
        MenuItem("Paste Ctrl+V"); menuFile.add(copyItem);
        menuFile.add(pasteItem);

        Menu menuHelp = new Menu("Help");
        MenuItem hTopicItem = new MenuItem("Help
        Topics"); MenuItem hAboutItem = new
        MenuItem("About Calculator");
        menuHelp.add(hTopicItem);
        menuHelp.addSeparator();
        menuHelp.add(hAboutItem);
```

```
        menu.add(menuFile);
        menu.add(menuHelp);

        fr.setMenuBar(menu);
        fr.setBounds(100, 100, 300, 200);
        fr.setTitle("Calculator");
        //fr.setResizable(false);
        fr.setVisible(true);

        // xử lý biến sự kiện đóng cửa số ứng dụng.
        fr.addWindowListener(
            new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                  System.exit(0);
                }
            });
      }
    }
```

Kết quả thực thi chương trình:

4.3.3. Gắn Component vào khung chứa

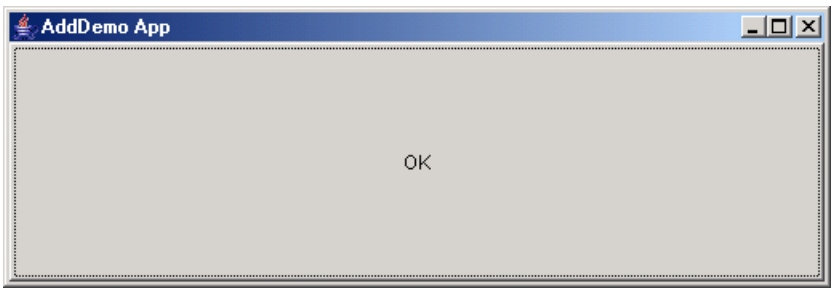
Để gắn một thành phần, một đối tượng component vào một cửa số (khung chứa) chúng ta dùng phương thức add của đối tượng khung chứa container.

***Ví dụ**:*

```
import java.awt.*;
class AddDemo
{
    public static void main(String args[])
    {
```

```
                    // Tạo đối tượng khung chứaFrame
                    Frame fr = new Frame("AddDemo
                    App");
// Tạo đối tượng Component
                    Button buttOk = new Button("OK");
                    // Gắn đối tượng nút nhấn vào khung chứa
                    fr.add(buttOk);
                    // Xác định kích thước, vị trí của Frame
                    fr.setSize(100, 100);
                    // Hiển thị Frame
                    fr.setVisible(true);
        }
    }
```

Kết quả thực thi chương trình:



4.3.4. Trình bày các Component trong khung chứa

- Như chúng ta đã biết khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗ ở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Chúng ta sẽ tìm hiểu chi tiết về các kiểu trình bày của thư viện AWT.

- Interface LayoutManager định nghĩa giao tiếp cho những lớp biết được làm thế nào để trình bày những trong những containers

4.3.4.1.FlowLayout

public class **FlowLayout** extends Object

implements LayoutManager, Serializable

Đối với một container trình bày theo kiểu FlowLayout thì:

- Các component gắn vào được sắp xếp theo thứ tự từ trái sang phải và từ trên xuống dưới.
- Các component có kích thước như mong muốn.
- Nếu chiều rộng của Container không đủ chỗ cho các component thì chúng tự động tạo ra một dòng mới.

- FlowLayout thường được dùng để để sắp xếp các button trong 1 panel.
- Chúng ta có thể điều chỉnh khoảng cách giữa các component.

***Ví dụ**:*

```
import java.awt.*;
import java.lang.Integer;
class FlowLayoutDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame("FlowLayout Demo");
        fr.setLayout(new FlowLayout());
        fr.add(new Button("Red"));
        fr.add(new Button("Green"));
        fr.add(new Button("Blue"));

        List li = new List(); for
        (int i=0; i<5; i++)
        {
            li.add(Integer.toString(i));
        }

        fr.add(li);
        fr.add(new Checkbox("Pick me", true));
        fr.add(new Label("Enter your name:"));
        fr.add(new TextField(20));
        // phương thức pack() được gọi sẽ làm cho cửa sổ
        // hiện hành sẽ có kích thước vừa với kích thước
        // trình bày bố trí những thành phần con của nó.
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:

4.3.4.2. BorderLayout

public class BorderLayout **extends** Object **implements** LayoutManager2, Serializable

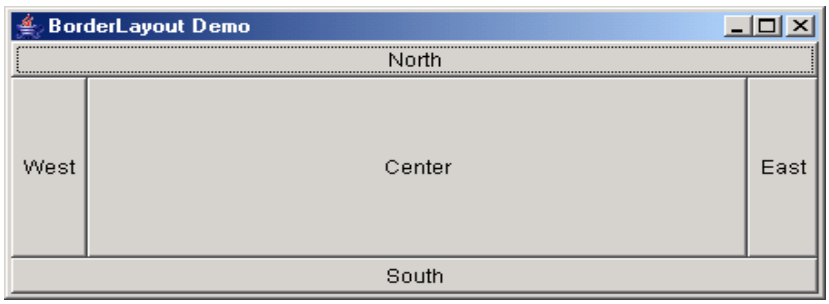
Đối với một container trình bày theo kiểu BorderLayout thì:

- Bộ trình bày khung chứa được chia làm 4 vùng: NORTH, SOUTH, WEST, EAST và CENTER. (Đông, Tây, Nam, Bắc và trung tâm). Bộ trình bày loại này cho phép sắp xếp và thay đổi kích thước của những components chứa trong nó sao cho vừa với 5 vùng ĐÔNG, TÂY, NAM, BẮC, TRUNG TÂM.
- Không cần phải gắn component vào cho tất cả các vùng.
- Các component ở vùng NORTH và SOUTH có chiều cao tùy ý nhưng có chiều rộng đúng bằng chiều rộng vùng chứa.
- Các component ở vùng EAST và WEST có chiều rộng tùy ý nhưng có chiều cao đúng bằng chiều cao vùng chứa.
- Các component ở vùng CENTER có chiều cao và chiều rộng phụ thuộc vào các vùng xung quanh.

***Ví dụ**:*

```
import java.awt.*;
class BorderLayoutDemo extends Frame
{
    private Button north, south, east, west, center;
    public BorderLayoutDemo(String sTitle)
    {
        super(sTitle);
        north  = new Button("North");
        south  = new Button("South");
        east   = new Button("East");
        west   = new Button("West");
        center  = new Button("Center");
        this.add(north, BorderLayout.NORTH);
        this.add(south, BorderLayout.SOUTH);
        this.add(east, BorderLayout.EAST);
        this.add(west, BorderLayout.WEST);
        this.add(center, BorderLayout.CENTER);
    }
    public static void main(String args[])
    {
        Frame fr = new BorderLayoutDemo ("BorderLayout
        Demo");
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.3.4.3. GridLayout

public class GridLayout extends **Object**
implements **LayoutManager**

Đối với một container trình bày theo kiểu GridLayout thì:

- Bộ trình bày tạo một khung lưới vô hình với các ô bằng nhau.
- Các đối tượng sẽ đặt vừa kích thước với từng ô đó. Thứ tự sắp xếp từ trái qua phải và từ trên xuống dưới

Ví dụ:

```
import java.awt.*;
public class GridLayoutDemo
{
        public static void main(String arg[])
        {
                Frame f = new Frame("GridLayout Demo");
                f.setLayout(new GridLayout(3,2));

                f.add(new Button("Red"));
                f.add(new Button("Green"));
                f.add(new Button("Blue"));
                f.add(new Checkbox("Pick me", true));
                f.add(new Label("Enter name here:"));
                f.add(new TextField());
                f.pack();
                f.setVisible(true);
        }
}
```

Kết quả thực thi chương trình:

4.3.4.4. GridBagLayout

public class **GridBagLayout** extends Object implements LayoutManager2
(public interface **LayoutManager2** extends LayoutManager)

Đối với một container trình bày theo kiểu GridBagLayout thì:

- Các componets khi được đưa vào khung chứa sẽ được trình bày trên 1 khung lưới vô hình tương tự như GridLayout. Tuy nhiên khác với GridLayout kích thước các đối tượng không nhất thiết phải vừa với 1 ô trên khung lưới mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉ định thông qua đối tượng GridBagConstraints.
- **Lớp GridBagConstraints** dẫn xuất từ lớp Object. Lớp GridBagConstraints dùng để chỉ định ràng buộc cho những components trình bày trong khung chứa container theo kiểu GridBagLayout.
  - **gridx, gridy:** vị trí ô của khung lưới vô hình mà ta sẽ đưa đối tượng con vào
  - **gridwidth, gridheight**: kích thước hay vùng trình bày cho đối tượng con.
  - **Insets:** là một biến đối tượng thuộc lớp Inset dùng để qui định khoảng cách biên phân cách theo 4 chiều (trên, dưới, trái, phải).
  - **weightx, weighty:** chỉ định khoảng cách lớn ra tương đối của các đối tượng con với nhau

4.3.4.5. Null Layout

- Một khung chứa được trình bày theo kiểu Null Layout có nghĩa là người lập trình phải tự làm tất cả từ việc qui định kích thước của khung chứa, cũng như kích thước và vị trí của từng đối tượng component trong khung chứa.
- Để thiết lập cách trình bày là Null Layout cho một container ta chỉ việc gọi phương thức **setLayout(null)** với tham số là **null.**
- Một số phương thức của lớp trừu tượng Component dùng để định vị và qui định kích thước của component khi đưa chúng vào khung chứa trình bày theo kiểu kiểu tự do:
  - Public void setLocation(Point p)
  - Public void setSize(Dimension p)
  - Public void setBounds(Rectangle r)

4.4.Các đối tượng khung chứa Container

- Như chúng ta đã biết container là đối tượng khung chứa có khả năng quản lý và chứa các đối tượng (components) khác trong nó.
- Các components chỉ có thể sử dụng được khi đưa nó vào 1 đối tượng khung chứa là container.
- Mỗi container thường gắn với 1 LayoutManager (FlowLayout, BorderLayout, GridLayout, GridBagLayout, Null Layout) qui định cách trình bày và bố trí các components trong một container.
- Các loại container trong java: Frame, Panel, Dialog, ScrollPanes.

4.4.1. Khung chứa Frame

java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--java.awt.Window
+--***java.awt.Frame***

- Khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.
- Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.

4.4.2Khung chứa Panel

java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--***java.awt.Panel***

- Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.
- Đối với khung chứa Panel thì các Panel có thể lồng vào nhau, vì vậy khung chứa Panel thường được dùng để bố trí các nhóm components bên trong một khung chứa khác.

4.4.3. Khung chứa Dialog

java.lang.Object
+--java.awt.Component
+--java.awt.Container
+--java.awt.Window
+--***java.awt.Dialog***

❖ Dialog là một lớp khung chứa tựa Frame và còn được gọi là popup window. Có hai loại dialog phổ biến:

- **Modal Dialog:** sẽ khóa tất cả các cửa số khác của ứng dụng khi dialog dạng này còn hiển thị
- **Non-Modal Dialog:** vẫn có thể đến các cửa số khác củaứng dụng khi dialog dạng này hiển thị.

❖ Một cửa sổ dạng Dialog luôn luôn phải gắn với một cửa sổ ứng dụng (Frame).
❖ Để tạo một đối tượng khung chứa Dialog ta có thể dùng một trong các constructor của nó:

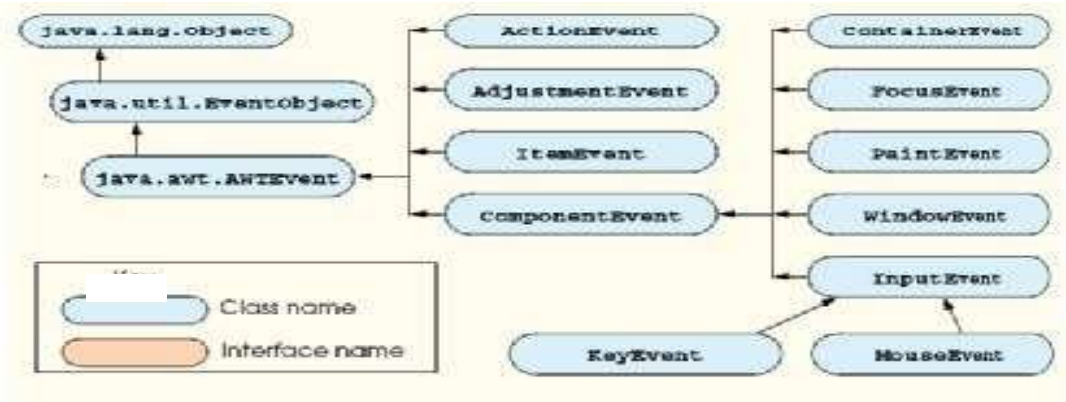
public Dialog (Frame parentWindow, boolean isModal) public Dialog (Frame parentWindow, String title, boolean isModal)
parentWindow: cửa sổ cha
title: tiêu đề của Dialog
isModal: true -> là Dialog dạng modal
isModal: false -> là Dialog không phải dạng modal (hay non-modal)

4.5. Xử lý biến cố - sự kiện

4.5.1. Mô hình xử lý sự kiện (Event-Handling Model)

- Ở trên chúng ta chỉ đề cập đến vấn đề thiết kế giao diện chương trình ứng dụng mà chưa đề cập đến vấn đề xử lý sự kiện. Những sự kiện được phát sinh khi người dùng tương tác với giao diện chương trình (GUI). Những tương tác thường gặp như: di chuyển, nhấn chuột, nhấn một nút nhấn, chọn một MenuItem trong hệ thống thực đơn, nhập dữ liệu trong một ô văn bản, đóng cửa sổ ứng dụng, … Khi có một tương tác xảy ra thì một sự kiện được gởi đến chương trình. Thông tin về sự kiện thường được lưu trữ trong một đối tượng dẫn xuất từ lớp AWTEvent. Những kiểu sự kiện trong gói java.awt.event có thể dùng cho cả những component AWT và JFC. Đối với thư viện JFC thì có thêm những kiểu sự kiện mới trong gói **java.swing.event.**

❖ ***Những lớp sự kiện của gói java.awt.event***



- Có 3 yếu tố quan trọng trong mô hình xử lý sự kiện:
  - Nguồn phát sinh sự kiện (event source)
  - Sự kiện (event object)
  - Bộ lắng nghe sự kiện (event listener)

✓ **Nguồn phát sinh sự kiện:** là thành phần của giao diện mà người dùng tác động.

✓ **Sự kiện:** Tóm tắt thông tin về xử kiện xảy ra, bao gồm tham chiếu đến

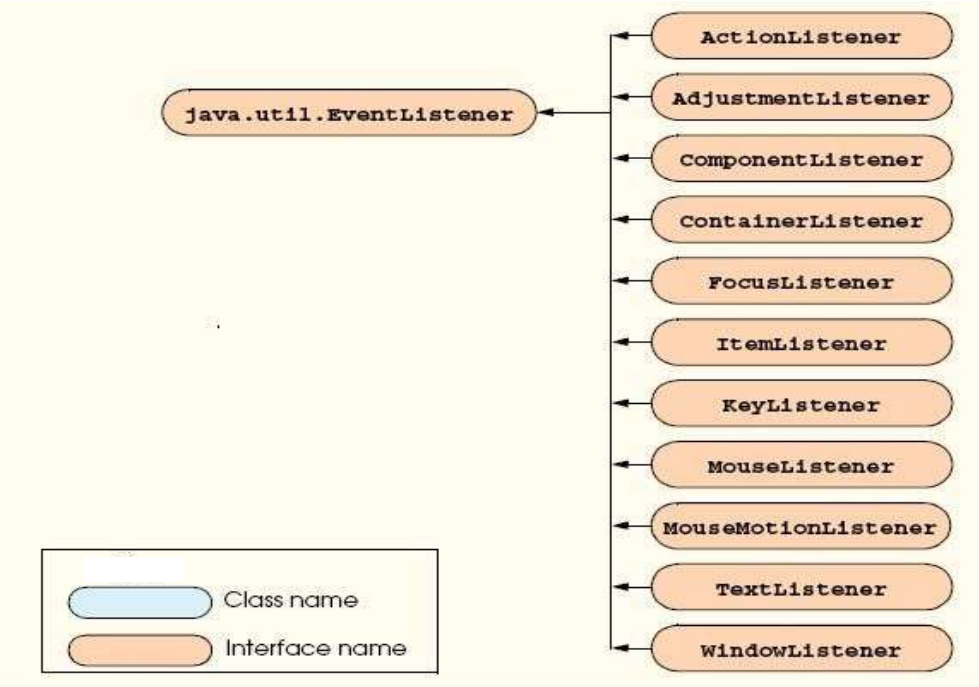
nguồn gốc phát sinh sự kiện và thông tin sự kiện sẽ gởi đến cho bộ lắng nghe xử lý.

✓ **Bộ lắng nghe:** Một bộ lắng nghe là một đối tượng của một lớp hiện thực một hay nhiều interface của gói java.awt.event hay java.swing.event (đối với những component trong thư viện JFC). Khi được thông báo, bộ lắng nghe nhận sự kiện và xử lý. Nguồn phát sinh sự kiện phải cung cấp những phương thức để đăng ký hoặc hủy bỏ một bộ lắng nghe. Nguồn phát sinh sự kiện luôn phải gắn với một bộ lắng nghe, và nó sẽ thông báo với bộ lắng nghe đó khi có sự kiện phát sinh đó.

➢ ***Như vậy người lập trình cần làm hai việc:***

- Tạo và đăng ký một bộ lắng nghe cho một component trên Gui
- Cài đặt các phương thức quản lý và xử lý sự kiện

Những interfaces lắng nghe của gói java.awt.event



- Một đối tượng **Event-Listener** lắng nghe những sự kiện khác nhau phát sinh từ các **components** của giao diện chương trình. Với mỗi sự kiện khác nhau phát sinh thì phương thức tương ứng trong những Event-Listener sẽ được gọi thực hiện.

- Mỗi **interface Event-Listener** gồm một hay nhiều các phương thức mà chúng cần cài đặt trong các lớp hiện thực (implements) interface đó. Những phương thức trong các interface là trừu tượng vì vậy lớp (bộ lắng nghe) nào hiện thực các interface thì phải cài đặt tất cả những phương thức đó. Nếu không thì các bộ lắng nghe sẽ trở thành các lớp trừu tượng.

4.5.2. Xử lý sự kiện chuột

Java cung cấp hai intefaces lắng nghe (bộ lắng nghe sự kiện chuột) là MouseListener và MouseMotionListener để quản lý và xử lý các sự kiện liên quan

đến thiết bị chuột. Những sự kiện chuột có thể "bẫy" cho bất kỳ component nào trên GUI mà dẫn xuất từ java.awt.component.

❖ **Các phương thức của interface MouseListener:**

- *public void mousePressed(MouseEvent event):* được gọi khi một nút chuột được nhấnvà con trỏ chuột ở trên component.
- *public void mouseClicked(MouseEvent event):* được gọi khi một nút chuột được nhấn và nhả trên component mà không di chuyển chuột.
- *public void mouseReleased(MouseEvent event):* được gọi khi một nút chuột nhả sa khi kéo rê.
- *public void mouseEntered(MouseEvent event):* được gọi khi con trỏ chuột vào trong đường biên của một component.
- *public void mouseExited(MouseEvent event)*: được gọi khi con trỏ chuột ra khỏi đường biên của một component.

❖ **Các phương thức của interface MouseMotionListener:**

- *public void mouseDragged(MouseEvent even):* phương thức này được gọi khi người dùng nhấn một nút chuột và kéo trên một component.
- *public void mouseMoved(MouseEvent event)*: phương thức này được gọi khi di chuyển chuột trên component.

- Mỗi phương thức xử lý sự kiện chuột có một tham số MouseEvent chứa thông tin về sự kiện chuột phát sinh chẳng hạn như: tọa độ x, y nơi sự kiện chuột xảy ra. Những phương thức tương ứng trong các interfaces sẽ tự động được gọi khi chuột tương tác với một component**.**

- Để biết được người dùng đã nhấn nút chuột nào, chúng ta dùng những phương thức, những hằng số của lớp InputEvent (là lớp cha của lớp MouseEvent).

4.5.3. Xử lý sự kiện bàn phím

- Để xử lý sự kiện bàn phím java hỗ trợ một bộ lắng nghe sự kiện đó là **interface *KeyListener***. Một sự kiện bàn phím được phát sinh khi người dùng nhấn và nhả một phím trên bàn phím. Một lớp hiện thực *KeyListener* phải cài đặt các phương thức *keyPressed, keyReleased* và *keyTyped*. Mỗi phương thức này có một tham số là một đối tượng kiểu *KeyEvent*. *KeyEvent* là lớp con của lớp *InputEvent*.

❖ **Các phương thức của interface *KeyListener***

- *Phương thức keyPressed:* được gọi khi một phím bất kỳ được nhấn.
- *Phương thức keyTyped:* được gọi thực hiện khi người dùng nhấn một phím không phải "phím hành động" (như phím mũi tên, phím Home, End, Page Up, Page Down, các phím chức năng như: Num Lock, Print Screen, Scroll Lock, Caps Lock, Pause).
- *Phương thức keyReleased:* được gọi thực hiện khi nhả phím nhấn sau khi sự kiện *keyPressed* hoặc *keyTyped*.

CHƯƠNG 5
LẬP TRÌNH CƠ SỞ DỮ LIỆU

5.1. Giới thiệu JDBC : (Java Database Connection)

Hầu hết các chương trình máy tính hiện này đếu ít nhiều liên quan đến việc truy xuất thông tin trong các cơ sở dữ liệu. Chính vì thế nên các thao tác hỗ trợ lập trình cơ sở dữ liệu là chức năng không thể thiếu của các ngôn ngữ lập trình hiện đại, trong đó có Java. JDBC API là thư viện chứa các lớp và giao diện hỗ trợ lập trình viên Java kết nối và truy cập đến các hệ cơ sở dữ liệu.

Phiên bản JDBC API mới nhất hiện nay là 3.0, là một thành phần trong J2SE, nằm trong 2 gói thư viện:

- **java.sql:** chứa các lớp và giao diện cơ sở của JDBC API.
- **javax.sql:** chứa các lớp và giao diện mở rộng. JDBC API cung cấp cơ chế cho phép một chương trình viết bằng Java có khả năng độc lập với các hệ cơ sở dữ liệu, có khả năng truy cập đến các hệ cơ sở dữ liệu khác nhau mà không cần viết lại chương trình. JDBC đơn giản hóa việc tạo và thi hành các câu truy vấn SQL trong chương trình.

➢ **Định nghĩa JDBC :**

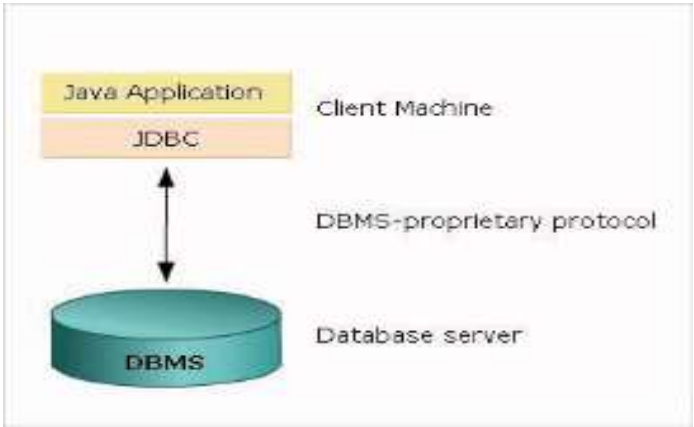
JDBC là API Java cơ sở, mà nó cung cấp một các lớp và các giao diện được viết bằng Java để truy xuất và thao tác với nhiều loại hệ cơ sở dữ liệu khác nhau. Sự kết hợp của JDBC API và Java nền tảng cung cấp các lợi thế cho việc truy xuất và bất kỳ nguồn dữ liệu khác nhau và sự linh hoạt của hoạt động trên một nền có hỗ trợ máy ảo Java (JVM). Đối với một nhà phát triển, đó là điều không cần thiết để viết một chương trình riêng biệt để truy cập vào các hệ cơ sở dữ liệu khác nhau như SQL Server, Oracle hoặc IBM DB2. Thay vào đó, một chương trình đơn lẻ với việc thực hiện JDBC có thể gửi Structured Query Language (SQL) hoặc gửi những câu lệnh khác tới các nguồn dữ liệu phù hợp hoặc hệ cơ sở dữ liệu.

➢ Những thuận lợi của JDBC :

- Sử dụng tiếp tục dữ liệu hiện có JDBC cho phép các ứng dụng doanh nghiệp tiếp tục sử dụng dữ liệu hiện có, ngay cả nếu dữ liệu được lưu trữ trên các hệ quản trị cơ sở dữ liệu khác nhau.
- Cung cấp độc lập Sự kết hợp của các Java API và JDBC API làm cho các cơ sở dữ liệu dịch chuyển từ một trong những nhà cung cấp này tới nhà cung cấp khác mà không cần các đoạn mã trong ứng dụng.
- Nền độc lập JDBC thường được sử dụng để kết nối với một ứng dụng người đến một "hậu trường" cơ sở dữ liệu, không có vấn đề của phần mềm quản lý cơ sở dữ liệu được sử dụng để kiểm soát các cơ sở dữ liệu. Trong vấn đề kiểu cách, JDBC là nền tảng chéo độc lập.
- Dễ sử dụng Với JDBC, sự phức tạp của một chương trình kết nối người dùng đến một "hậu trường" cở sở dữ liệu bị ẩn đi, và làm cho nó dễ dàng triển khai hơn, kinh tế hơn để duy trì.

5.2. Kiến trúc JDBC :

➢ <u>Mô hình hai tầng JDBC :</u>

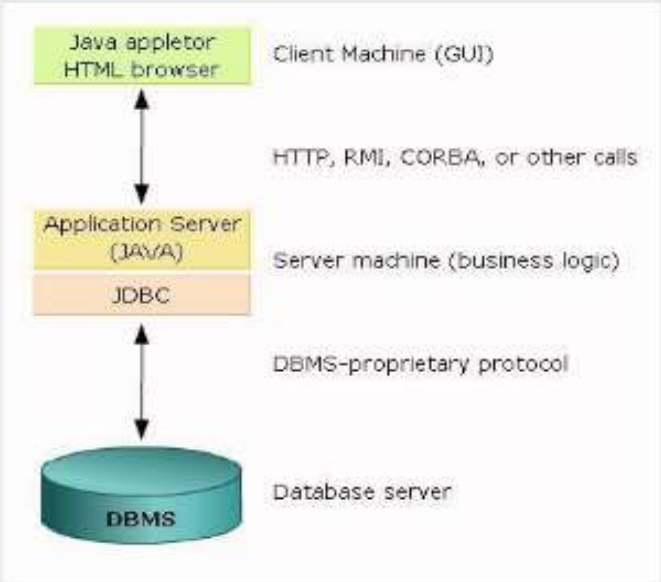


- API hỗ trợ trợ mô hình hai tầng cũng như mô hình ba tầng xử lý dữ liệu cho các mô hình truy xuất cơ sở dữ liệu.

- Trong mô hình hai tầng hệ thống máy khách / máy chủ: máy khách có thể liên hệ trực tiếp với cơ sở dữ liệu của máy chủ mà không cần của bất kỳ một công nghệ trung gian hoặc máy chủ khác.

-Trong mô hình hai tầng môi trường JDBC, các ứng dụng Java là khách và DBMS là cơ sở dữ liệu máy chủ. Việc thực hiện tiêu biểu của mô hình hai tầng liên quan đến việc sử dụng JDBC API để chuyển và gửi yêu cầu của khách hàng tới cơ sở dữ liệu. Cơ sở dữ liệu có thể nằm cùng trên một mạng hoặc có khác mạng. Các kết quả được gửi trả về cho khách hàng một lần nữa thông qua JDBC API.

- Mô hình ba tầng :



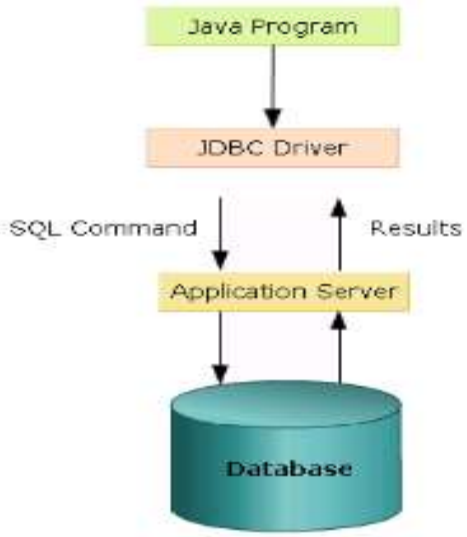
- Trong mô hình ba tầng: tầng giữa là tầng các dịch vụ, một máy chủ thứ ba đảm nhiệm việc gửi yêu cầu của khách hàng tới máy chủ cơ sở dữ liệu. Tầng giữa giúp việc tách các cơ sở dữ liệu máy chủ từ máy chủ WEB. Sự tham gia của máy chủ thứ 3 hoặc máy chủ Proxy tăng cường an nình bằng cách đi qua tất cả các yêu cầu đến máy chủ cơ sở dữ liệu thông qua máy chủ Proxy. Máy chủ cơ sở dữ liệu xử lý các yêu cầu và gửi lại các kết quả đến tầng giữa (Proxy Server), một lần nữa kết quả được gửi trả về máy khách (Client).

- Mô hình ba tầng có lợi thế hơn so với mô hình hai tầng là nó đơn giản hóa hơn và giảm chi phí triển khai ứng dụng, ngoài ra nó còn cung cấp và sửa đổi quyền truy xuất vào cơ sở dữ liệu.

- **JDBC API :**

- JDBC API là bộ sưu tập của các cách định nghĩa cơ sở dữ liệu theo nhiều cách khác nhau và các ứng dụng giao tiếp với nhau.

- Cốt lõi của JDBC API được dựa trên Java, vì vậy, nó được dùng như là nền tảng để xây dựng chung giữa ba tầng kiến trúc. Do đó, JDBC API là tầng giữa. Nó định nghĩa thế nào là mở kết nối của một ứng dụng và cơ sở dữ liệu, các yêu cầu được gửi tới cơ sở dữ liệu, các câu lệnh truy vấn SQL được thực thi, và kết quả của câu truy vấn đó được lấy ra, JDBC đã đạt được mục tiêu thông qua một tập các giao diện Java, đó là sự thực hiện một cách riêng biết của một lớp cho một cơ sở dữ liệu cụ thể và được gọi là trình điều khiển JDBC (JDBC Driver).



5.3. Các khái niệm cơ bản

5.3.1. JDBC Driver :

❖ Để có thể tiến hành truy cập đến các hệ quản trị cơ sở dữ liệu sử dụng kỹ thuật JDBC, chúng ta cần phải có trình điều khiển JDBC của hệ quản trị CSDL mà chúng ta đang sử dụng. Trình điều khiển JDBC là đoạn chương trình, do chính nhà xây dựng hệ quản trị CSDL hoặc do nhà cung ứng thứ ba cung cấp, có khả năng yêu cầu hệ quản trị CSDL cụ thể thực hiện các câu lệnh SQL.

❖ Danh sách các trình điều khiển JDBC cho các hệ quản trị CSDL khác nhau được Sun cung cấp và cập nhật liên tục tại địa chỉ :

http://industry.java.sun.com/products/jdbc/drivers.

❖ **Các trình điều khiển JDBC được phân làm 4 loại khác nhau :**

▪ **Loại 1: có tên gọi là *Bridge Driver*** : Trình điều khiển loại này kết nối với các hệ CSDL thông qua cầu nối ODBC. Đây chính là chình điều khiển được sử dụng phổ biến nhất trong những ngày đầu Java xuất hiện. Tuy nhiên, ngày nay trình điều khiển loại này không còn phổ biến do có nhiều hạn chế. Trình điều khiển loại này luôn được cung cấp kèm trong bộ J2SE với tên: **sun.jdbc.odbc.JdbcOdbcDriver.**

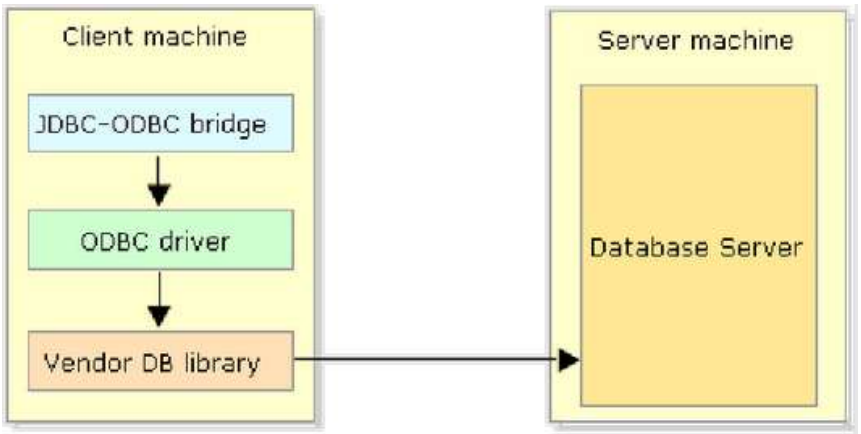


Figure 1. Type 1: JDBC-ODBC Bridge

▪ **Loại 2: có tên gọi là *Native API Driver*.**

Trình điều khiển loại này sẽ chuyển các lời gọi của JDBC API sang thư viện hàm (API) tương ứng với từng hệ CSDL cụ thể. Trình điều khiện loại này thường chỉ do nhà xây dựng hệ CSDL cungncấp. Để có thể thi hành chương trình mã lệnh để làm việc với hệ CSDL cụ thể cần phải được cung cấp đi kèm với chương trình.

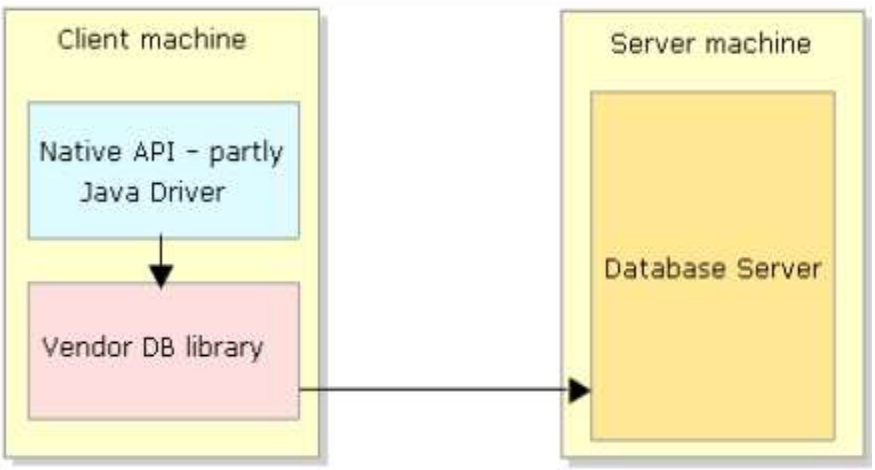


Figure 2. Type 2: Native-API/partly Java driver

- **Loại 3: có tên gọi là *JDBC-Net Driver*** :Trình điều khiển loại này sẽ chuyển các lời gọi JDBC API sang một dạng chuẩn độc lập với các hệ CSDL, và sau được chuyển sang lời gọi của hệ CSDL cụ thể bởi 1 chương trình trung gian. Trình điều khiển của các nhà cung ứng thứ 3 thường thuộc loại này. Lợi thế của trình điều khiển loại này là không cần cung cấp mã lệnh kèm theo và có thể sử dụng cùng một trình điều khiển để truy cập đến nhiều hệ CSDL khác nhau.

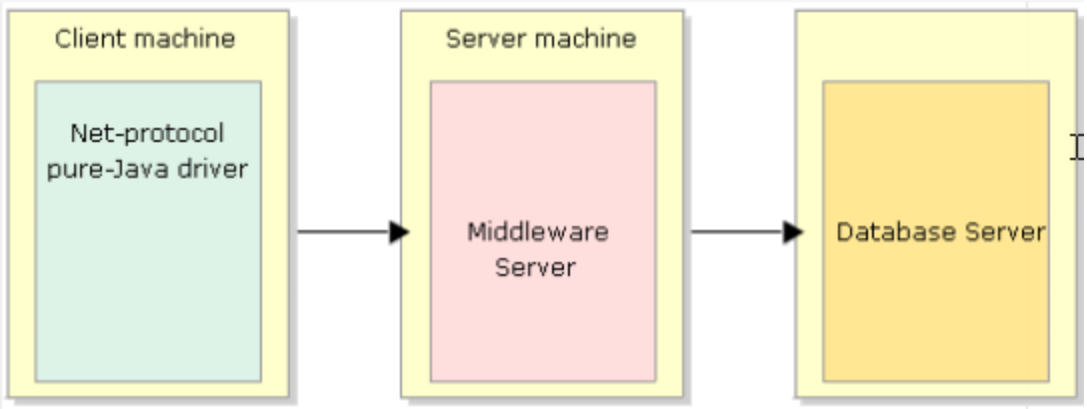


Figure 3. Type 3: Net-protocol/all Java driver

- **Loại 4: có tên gọi là *Native Protocol Driver*.**

Trình điều khiển loại này chuyển các lời gọi JDBC API sang mã lệnh của hệ CSDL cụ thể. Đây là các trình điều khiển thuần Java, có nghĩa là không cần phải có mã lệnh của hệ CSDL cụ thể khi thi hành chương trình.

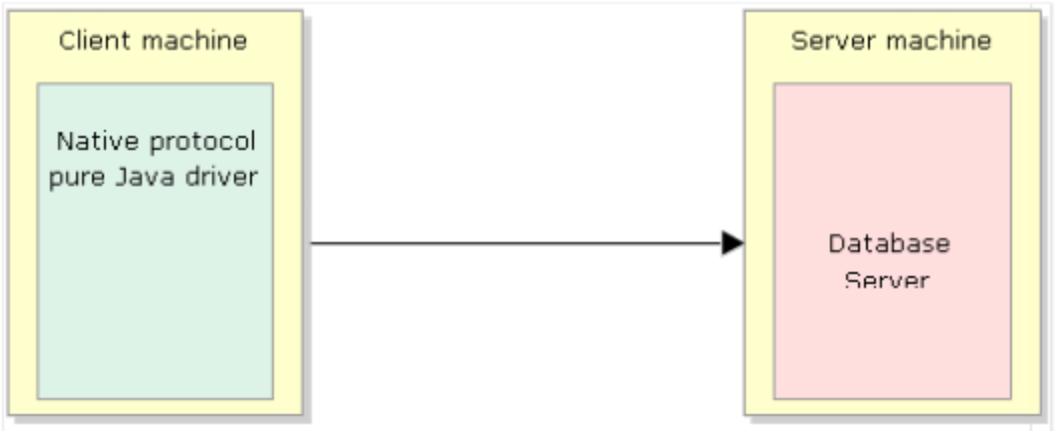


Figure 4. Type 4: Native-protocol/all Java driver

5.3.2. JDBC URL

Để có thể kết nối với CSDL, chúng ta cần xác định nguồn dữ liệu cùng với các thông số liên quan dưới dạng 1 URL như sau: ***jdbc:<subprotocol>:<dsn>:<others>***

Trong đó:

- <***subprotocol***>: được dùng để xác định trình điều khiển để kết nối với CSDL.
- <***dsn***>: địa chỉ CSDL. Cú pháp của <***dsn***> phụ thuộc vào từng trình điều khiển cụ thể.
- <***other***>: các tham số khác

Ví dụ: *jdbc:odbc:dbname* là URL để kết nối với CSDL tên dbname sử dụng cầu nối ODBC.

jdbc:microsoft:sqlserver://hostname:1433 là URL để kết nối với CSDL Microsoft SQL Server.Trong đó hostname là tên máy cài SQL Server.

5.4. Kết nối CSDL với JDBC

Việc kết nối với CSDL bằng JDBC được thực hiện qua hai bước: đăng ký trình điều khiển JDBC; tiếp theo thực thi phương thức:

getConnection() của lớp DriverManager.

5.4.1. Đăng ký trình điều khiển

Trình điều khiển JDBC được nạp khi mã bytecode của nó được nạp vào JVM. Một cách đơn giản để thực hiện công việc này là thực thi phương thức:

Class.forName("<JDBC Driver>").

Ví dụ: để nạp trình điều khiển sử dụng cầu nối ODBC do Sun cung cấp, chúng ta sử dụng câu lệnh sau :

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").

5.4.2. Thực hiện kết nối

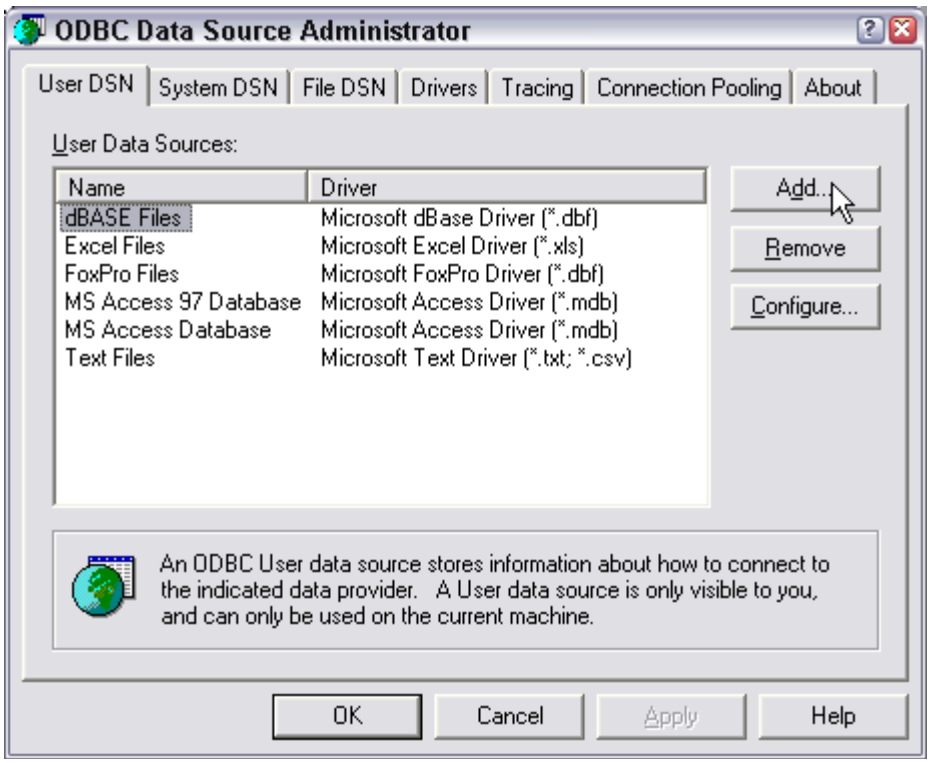
Sau khi đã nạp trình điều khiển JDBC, việc kết nối với CSDL được thực hiện với một trong các phương thức sau trong lớp DriverManager:

- **public static Connection getConnection(String url) throws SQLException**: thực hiện kết nối với CSDL được yêu cầu. Bộ quản lý trình điều khiển sẽ tự động lựa chọn trình điều khiển phù hợp trong số các trình điều khiển đã được nạp.
- **public static Connection getConnection(String url,String user, String pass) throws SQLException:** tiến hành kết nối tới CSDL với tài khoản user và mật mã pass.

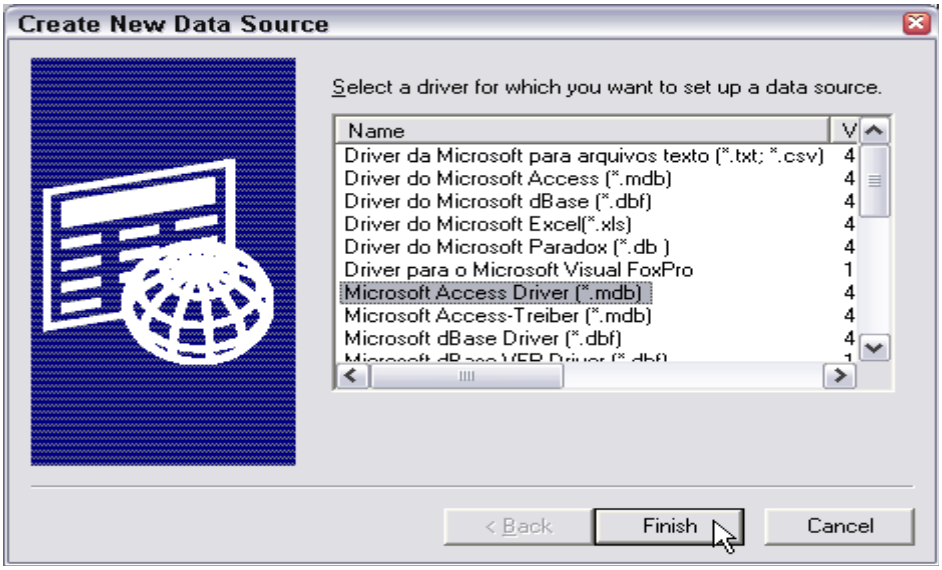
- **public static Connection getConnection(String url,Properties info) throws SQLException:** tương tự hai phương thức trên ngoài ra cung cấp thêm các thông tin qui định thuộc tính kết nối thông qua đối tượng của lớp Properties.Kết quả trả về của các phương thức trên là một đối tượng của lớp java.sql.Connection được dùng để đại diện cho kết nối đếnCSDL.

5.4.3. Ví dụ

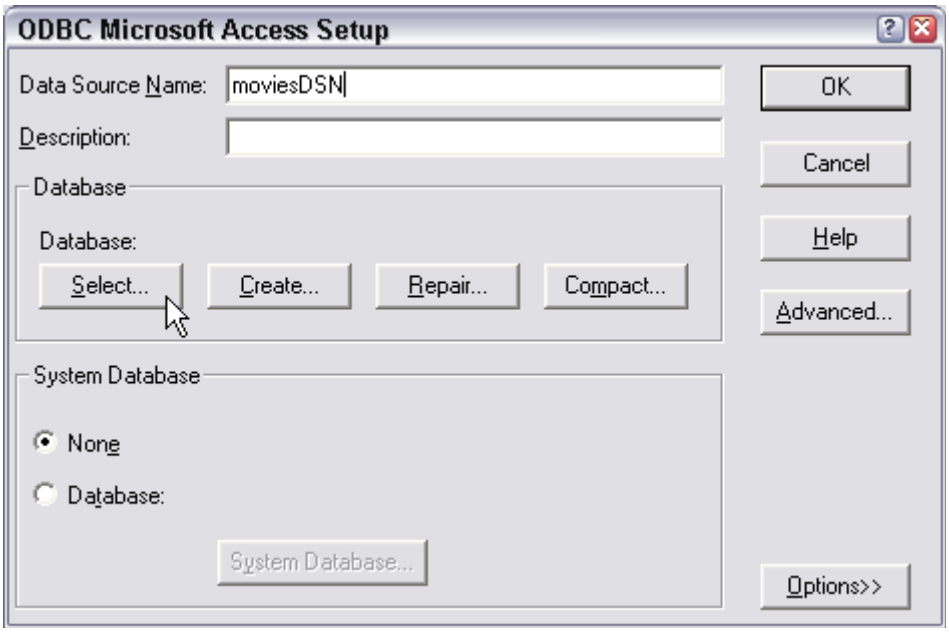
Trong phần ví dụ này chúng ta sẽ tìm hiểu các cách khác nhau để kết nối với tập tin CSDl Access movies.mdb có một bảng tên Movies. Bảng này gồm các cột number, title, category và fomat.Để có thể tiến hành kết nối với Microsoft Access thông qua cầu nối ODBC sau khi đã tạo tập tin CSDL movies.mdb, chúng ta cần phải tạo Data Source Name cho CSDL bằng cách vào Control Panel và chọn ODBC Data Source.



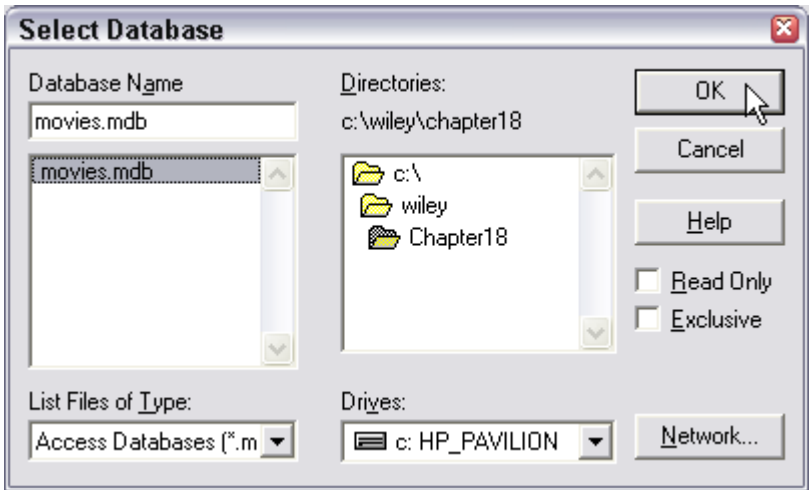
Tiếp theo nhấn vào nút Add, bạn sẽ thấy hiển thị danh sách các trình điều khiển CSDL hiện có.



Bạn chọn Microsoft Access Driver(*.mdb) và nhấn Finish. Cửa sổ cấu hình cho tập tin Access sẽ xuất hiện và nhập moviesDSN vào ô Data Source Name



Bạn nhấn nút Select và chọn tập tin CSDL cần tạo data source name. Sau đó nhấn OK để kết thúc.



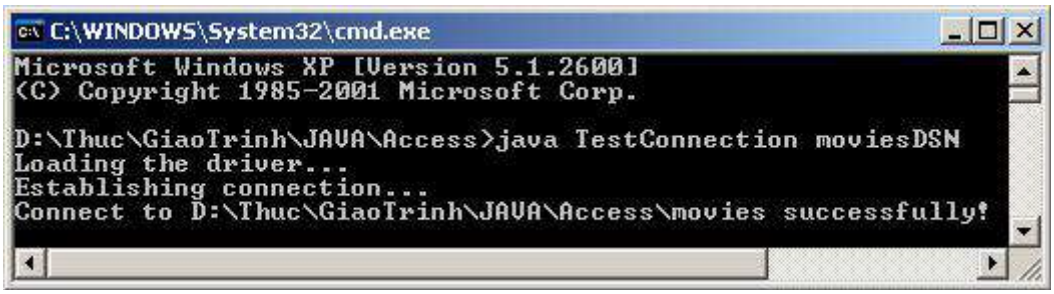
Sau khi đã hoàn tất công việc tạo DSN cho tập tin movies.mdb,chúng ta có thể sử dụng đoạn mã sau để tiến hành kết nối với tập tin movies.mdb.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class TestConnection{
public static void main(String args[]){
        Connection connection = null;
if( args.length != 1) {
    System.out.println("Syntax: java TestConnection " +"DSN");
    return;
}
try { // load driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    System.out.println("Loading the driver...");
}
catch( Exception e ) { //problem load driver,class notexist
e.printStackTrace( );
return;
}
try {
```

```
        String dbURL = "jdbc:odbc:" + args[0];
        System.out.println("Establishing connection...");
        connection =DriverManager.getConnection(dbURL,"","");
        System.out.println("Connect to " +connection.getCatalog() + " successfully!");
        // Do whatever queries or updates you want here!!!
}
catch( SQLException e ) {
e.printStackTrace( );
}
finally {
            if( connection != null ) {
               try { connection.close( ); }
               catch( SQLException e ) {
                     e.printStackTrace( );
               }
            }
         }
}
}
```

Sau khi biên dịch đoạn chương trình trên, chúng ta có thể thực hiện kết nối với CSDL bằng cách thực thi câu lệnh:

java TestConnection moviesDSN



5.5. Kiểu dữ liệu SQL và kiểu dữ liệu JaVa

Trong quá trình thao tác với CSDL, chúng ta sẽ gặp phải vấn đề chuyển đổi giữa kiểu dữ liệu trong CSDL sang kiểu dữ liệu Java hỗ trợ và ngược lai. Việc chuyển đổi này được thực hiện như trong 2 bảng sau:

SQL Type	Java Type
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	Int
BIGINT	Long
REAL	Float
FLOAT	Double
DOUBLE	Double
DECIMAL	java.math.BigDecimal
NUMERIC	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
LONGVARCHAR	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BINARY	byte[ ]
VARBINARY	byte[ ]
LONGVARBINARY	byte[ ]
BLOB	java.sql.Blob
CLOB	Java.sql.Clob
ARRAY	Java.sql.Array
REF	Java.sql.Ref
STRUCT	Java.sql.Struct

Bảng chuyển đổi từ kiểu dữ liệu SQL sang Java

Java Type	SQL Type
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
java.lang.String	VARCHAR or LONGVARCHAR
byte[ ]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Blob	BLOB
java.sql.Clob	CLOB
java.sql.Array	ARRAY
java.sql.Ref	REF
java.sql.Struct	STRUCT

Bảng chuyển đổi từ kiểu dữ liệu Java sang SQL

5.6. Các thao tác cơ bản trên CSDL

- Các thao tác truy vấn CSDL chỉ có thể được thực hiện sau khi đã có đối tượng Connection, được tạo ra từ quá trình kết nối vào CSDL. Chúng ta sử dụng đối tượng của lớp Connection để tạo ra các thể hiện của lớp java.sql.Statement. Sau khi tạo ra các đối tượng của lớp Statement chúng ta có thể thực hiện các thao tác trong các đối tượng statement trên connection tương ứng.

- Nội dung trong một statement chính là các câu SQL. Câu lệnh SQL trong các statement chỉ được thực hiện khi chúng ta gửi chúng đến CSDL. Nếu câu lện SQL là một câu truy vấn nội dung thì kết quả trả về sẽ là một thể hiện của lớp java.sql.ResultSet, ngược lại (các câu lệnh thay đổi nội dung CSDL) sẽ trả về kết quả là mộ số nguyên. Các đối tượng của lớp ResultSet cho phép chúng ta truy cập đến kết quả trả về của các câu truy vấn.

5.6.1.Các lớp cơ bản

- ***java.sql.Statement :***

- **Statement** là một trong 3 lớp JDBC cơ bản dùng để thể hiện một câu lệnh SQL. Mọi thao tác trên CSDL được thực hiện thông qua 3 phương thức của lớp Statement.

- **Phương thức executeQuery()** nhận vào 1 tham số là chuỗi nội dung câu lện SQL và trả về 1 đối tượng kiểu ResultSet. Phương thức này được sử dụng trong các trường hợp câu lệnh SQL có trả về các kết quả trong CSDL.

-**Phương thức executeUpdate()** cũng nhận vào 1 tham số là chuỗi nội dung câu lệnh SQL. Tuy nhiện phương thức này chỉ sử dụng được đối với các cây lệnh cập nhật nội dung CSDL. Kết quả trả về là số dòng bị tác động bởi câu lệnh SQL.Phương thức execute() là trường hợp tổng quát của 2 phương thức trên. Phương thức nhận vào chuỗi nội dung câu lệnh SQL. Câu lệnh SQL có thể là câu lệnh truy vấn hoặc cập nhật. Nếu kết quả của câu lệnh là các dòng trong CSDL thì phương thức trả về giá trị true, ngược lại trả về giá trị false.

+ Trong trường hợp giá trị true, sau đó chúng ta có thể dùng phương thức getResultSet() để lấy các dòng kết quả trả về.

- ***java.sql.ResultSet***

- Đối tượng resultset là các dòng dữ liệu trả về của câu lệnh truy vấn CSDL. Lớp này cung cấp các phương thức để rút trích các cột trong từng dòng kết quả trả về. Tất cả

các phương thức này đều có dạng: **type** get**Type**(int | String)

- Trong đó tham số có thể là số thứ tự của cột hoặc tên cộtcần lấy nội dung.Tại 1 thời điểm chúng ta chỉ có thể thao tác trên 1 dòng của resultset. Để thao tác trên dòng tiếp theo chúng ta sử dụng phương thức next(). Phương thức trả về giá trị true trong trường hợp có dòng tiếp theo, ngược lại trả về giá trị fale

5.6.2.Ví dụ truy vấn CSDL

```
public class Movie{
private String movieTitle, category, mediaFormat;
private int number;
public Movie(int n, String title, String cat, String format){
        number = n;
        movieTitle = title;
        category = cat;
        mediaFormat = format;
}
public int getNumber(){return number;}
public String getMovieTitle(){return movieTitle;}
public String getCategory(){return category;}
public void setCategory(String c){category = c;}
public String getFormat(){return mediaFormat;}
public void setFormat(String f){mediaFormat = f;}
public String toString(){
    return number + ": " + movieTitle + " - " + category + " " + mediaFormat;
   }
}

import java.sql.*;
public class MovieDatabase{
private Connection connection;
private PreparedStatement findByNumber, updateCategory;
private CallableStatement findByCategory;
```

```
public MovieDatabase(Connection connection) throws SQLException{
                this.connection = connection;
}
public void showAllMovies(){
try{
    Statement selectAll = connection.createStatement();
    String sql = "SELECT  * FROM Movies";
    ResultSet results = selectAll.executeQuery(sql);
    while(results.next()){
    int number = results.getInt(1);
    String title = results.getString("title");
    String category = results.getString(3);
    String format = results.getString(4);
    Movie movie = new Movie(number, title, category,
    format);
    System.out.println(movie.toString());
}
results.close();
selectAll.close();
}
catch(SQLException e){
                        e.printStackTrace();
                    }
}
}
import java.sql.*;
public class ShowMovies{
public static void main(String [] args){
String url = "jdbc:odbc:" + args[0];
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    Connection connection =
    DriverManager.getConnection(url);
    MovieDatabase db = new
    MovieDatabase(connection);
    db.showAllMovies();
    connection.close();
  }
catch(Exception e){
                    e.printStackTrace();
                   }
}
}
```

5.6.3.Ví dụ cập nhật CSDL

Phương thức addMovie() bên dưới được thêm vào lớp MovieDatabase đã định nghĩa ở ví dụ trên.

```
public class MovieDatabase{
...
public void addMovie(Movie movie){
System.out.println("Adding movie: " + movie.toString());
try{
    Statement addMovie = connection.createStatement();
String sql = "INSERT INTO Movies VALUES(" + movie.getNumber() + ", "
                                          + "'" + movie.getMovieTitle() + "', "
                                          + "'" + movie.getCategory()  + "', "
                                          + "'" + movie.getFormat()    + "')";
    System.out.println("Executing statement: " + sql);
    addMovie.executeUpdate(sql);
    addMovie.close();
    System.out.println("Movie added successfully!");
}
```

```
catch(SQLException e){
e.printStackTrace();
}
}
}
import java.sql.*;
public class AddMovies{
public static void main(String [] args){
String url = "jdbc:odbc:" + args[0];
System.out.println("Attempting to connect to " + url);
try{
    System.out.println("Loading the driver...");
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    System.out.println("Establishing a connection...");
    Connection connection = DriverManager.getConnection(url);
    System.out.println("Connect to "+ connection.getCatalog() + " a success!");
    MovieDatabase db = new MovieDatabase(connection);
    Movie [] movies = new Movie[6];
    movies[0] = new Movie(1, "Star Wars: A New Hope","Science Fiction",
    "DVD");
    movies[1] = new Movie(2, "Citizen Kane", "Drama","VHS");
    movies[2] = new Movie(3, "The Jungle Book","Children", "VHS");
    movies[3] = new Movie(4, "Dumb and Dumber","Comedy", "DVD");
    movies[4] = new Movie(5, "Star Wars: Attack of theClones", "Science Fiction",
    "DVD");
    movies[5] = new Movie(6, "Toy Story", "Children","DVD");
for(int i = 0; i < movies.length; i++){
    db.addMovie(movies[i]);
}
  System.out.println("Closing the connection...");
  connection.close();
```

```
}
catch(Exception e){
    e.printStackTrace();
}
}
}
```

CHƯƠNG 6

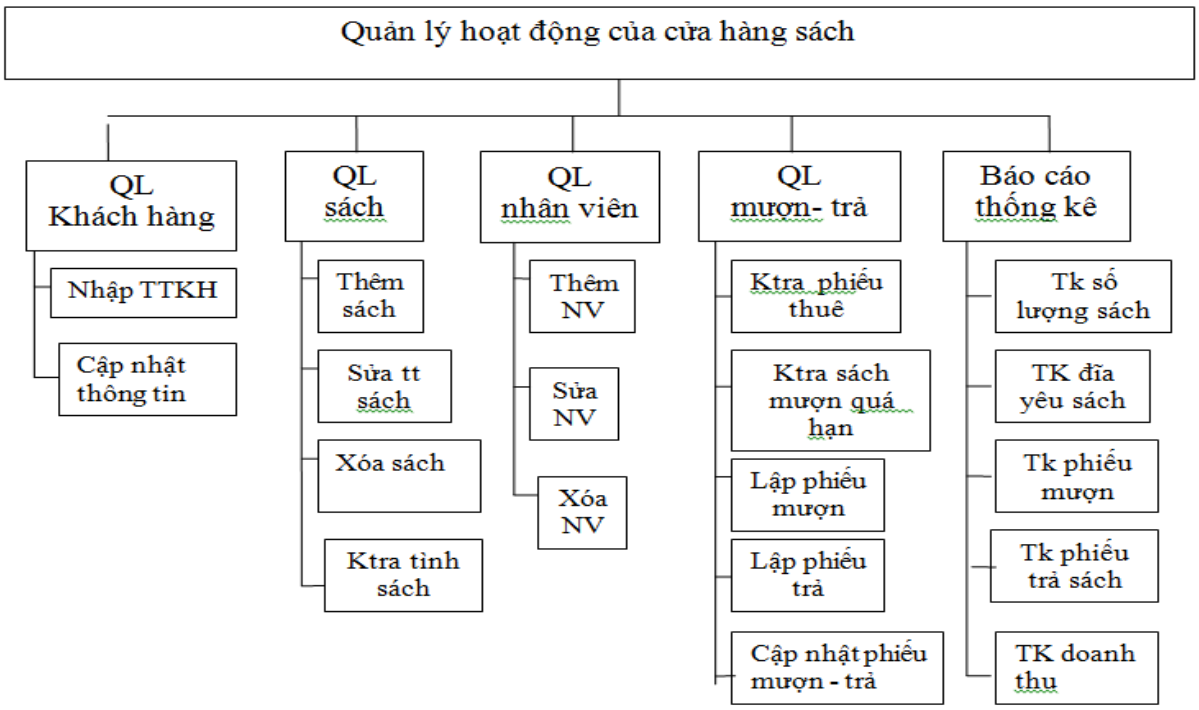
PHẦN MỀM ỨNG DỤNG QUẢN LÝ CỬA HÀNG SÁCH

6.1. Bài toán dặt ra

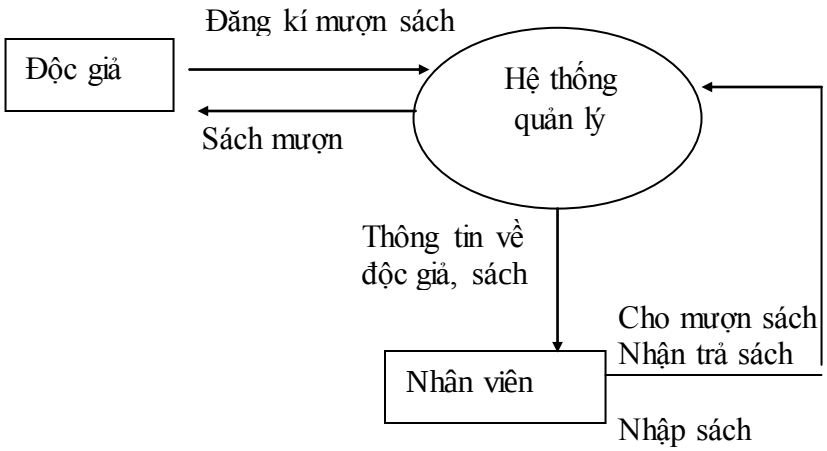
- Ngày nay, cùng với sự phát triển của khoa học kĩ thuật, nhu cầu ứng dụng tin học trong công tác quản lý cũng ngày càng gia tăng.Việc xây dựng các phần mềm quản lý nhằm đáp ứng nhu cầu trên là cần thiết.
- Trong một cửa hàng cho thuê sách có quá nhiều thông tin để quản lý, (VD: khách hàng, thuê đĩa trả sách,….). Do vậy sẽ dẫn đến sự quá tải thông tin
- Bạn sẽ làm gì khi không thể kiểm soát và quản lý????
- Giải pháp: phần mềm hệ thống quản lý của hàng sách sẽ giúp bạn xử lý và lưu thông tin dễ dàng hơn

6.2. Phân tích chức năng của hệ thống

Biểu đồ phân cấp chức năng

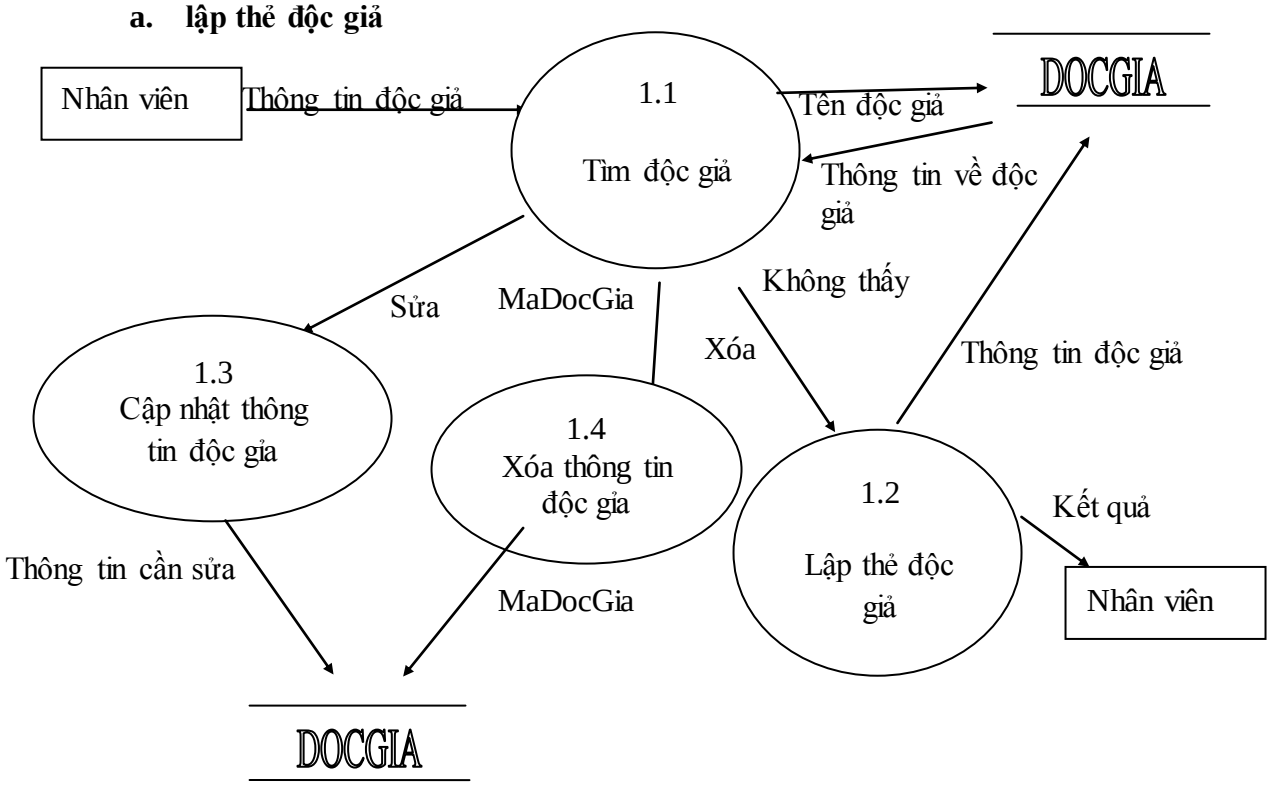


- **Mô hình mức đỉnh**
  - mức 0

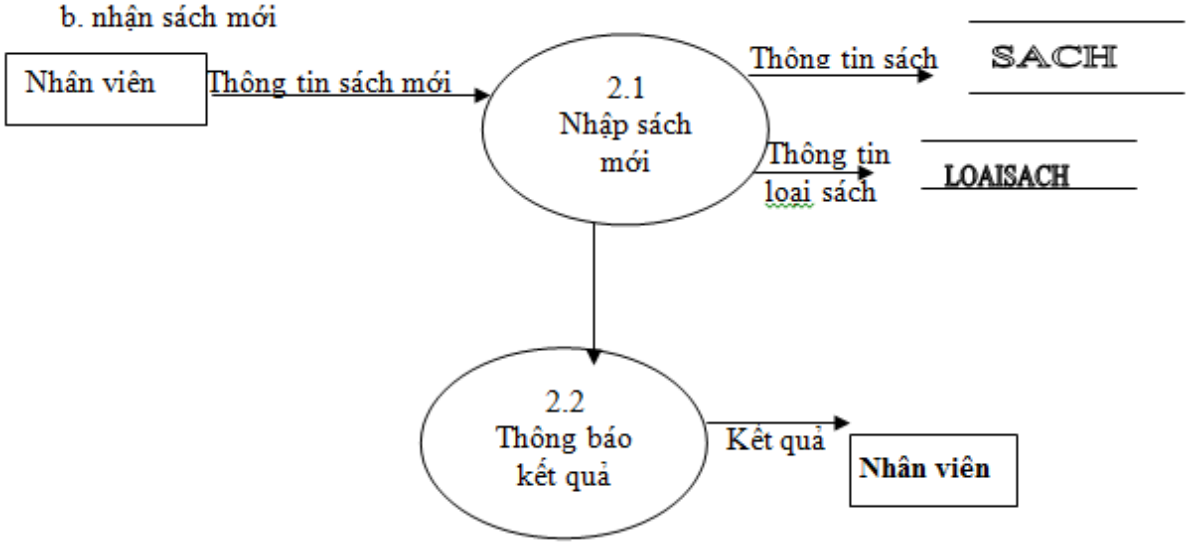


- mức 1

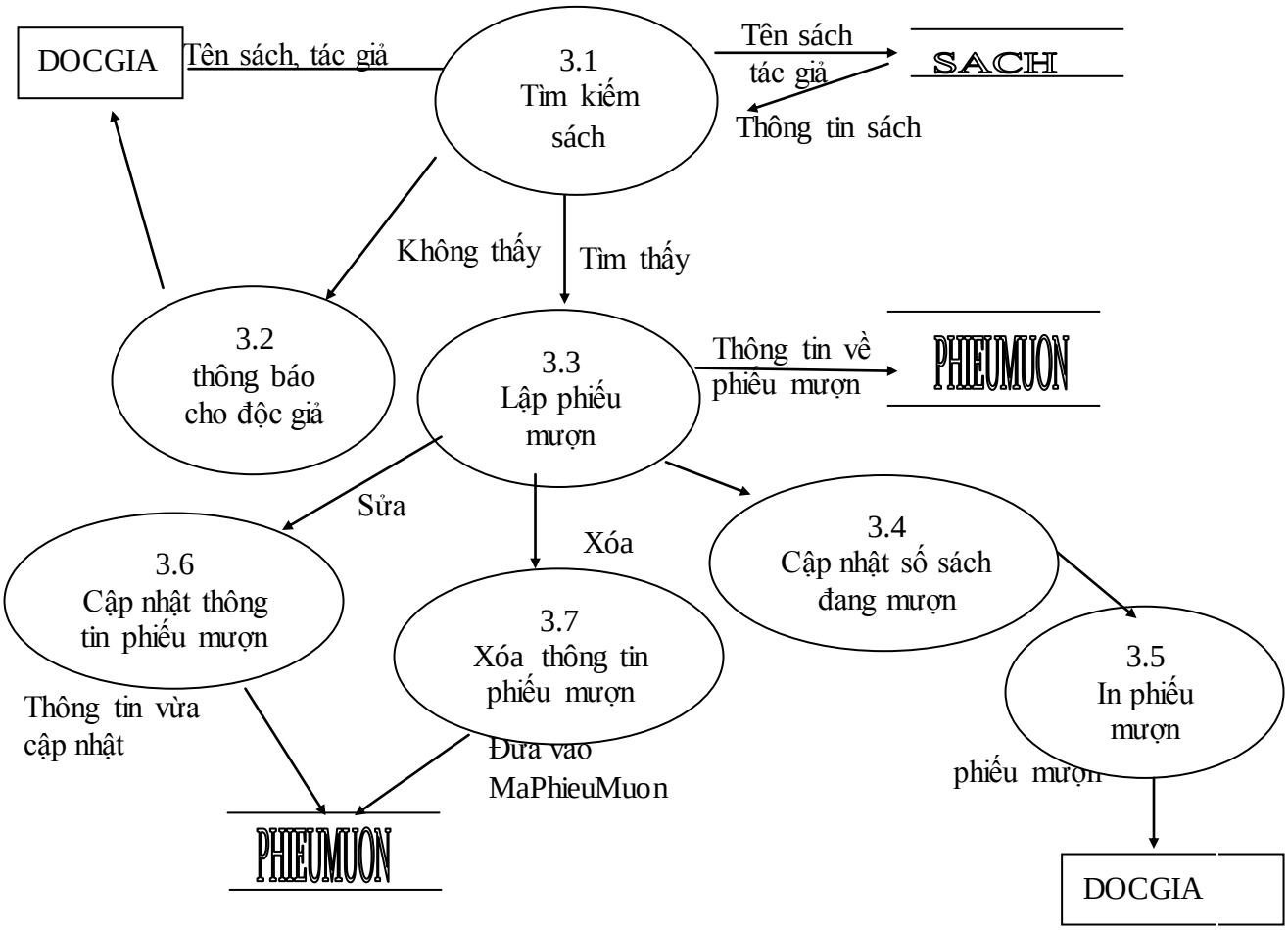
a. lập thẻ độc giả



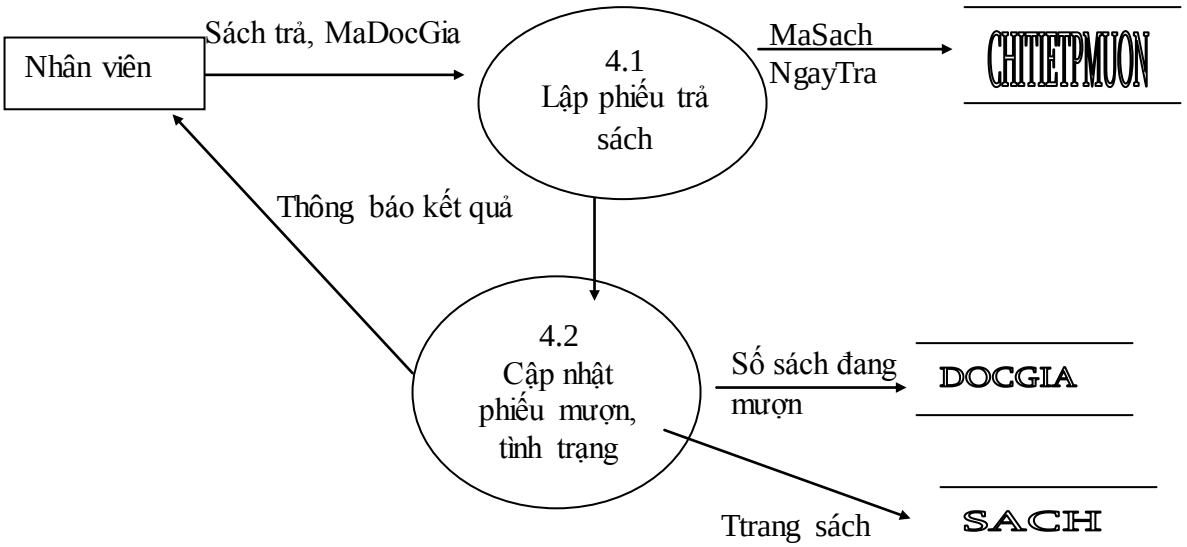
b. nhận sách mới



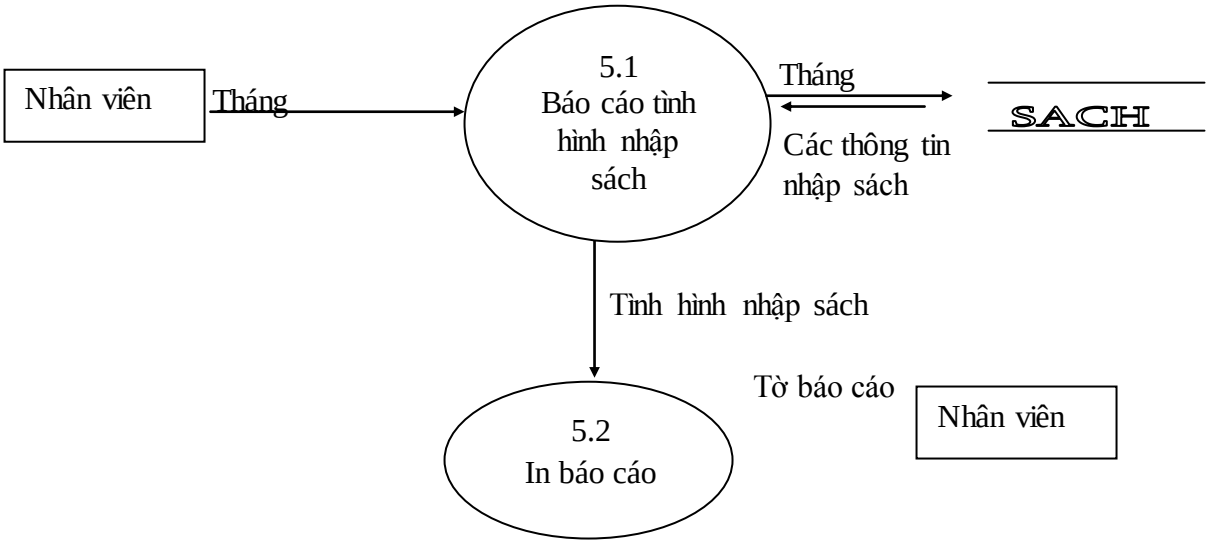
c. Lập phiếu mượn



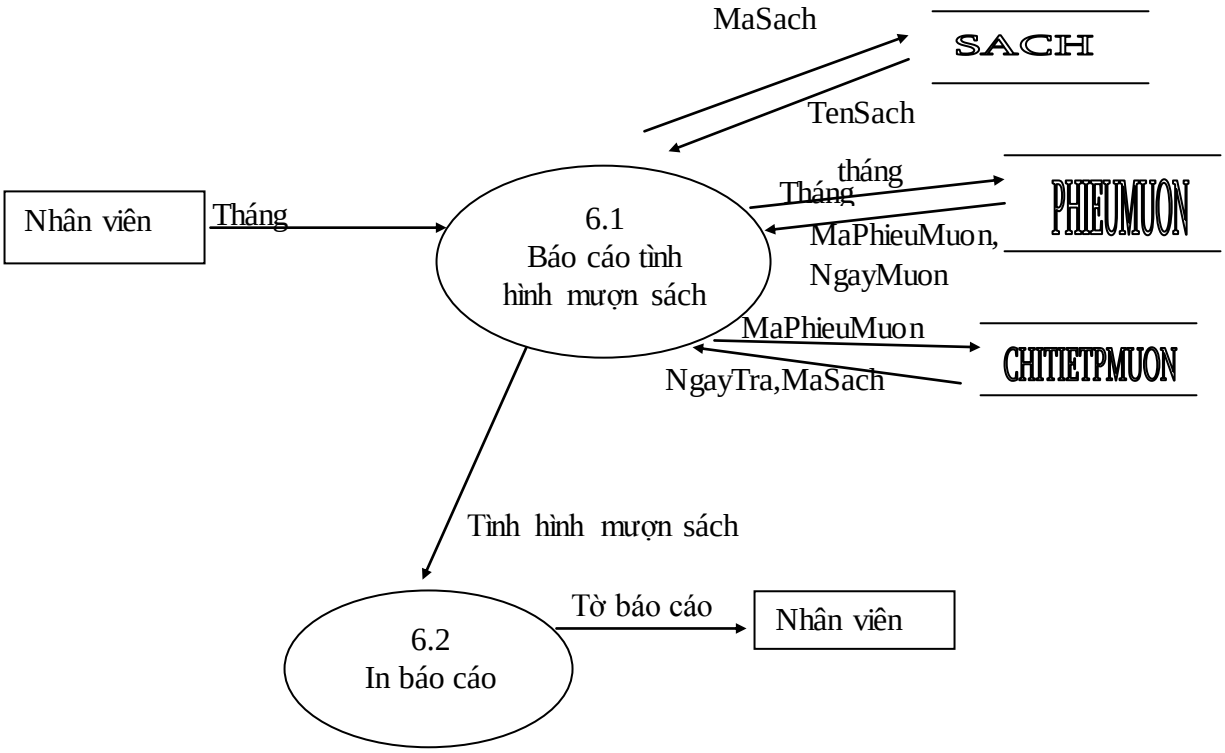
d. Nhận trả sách



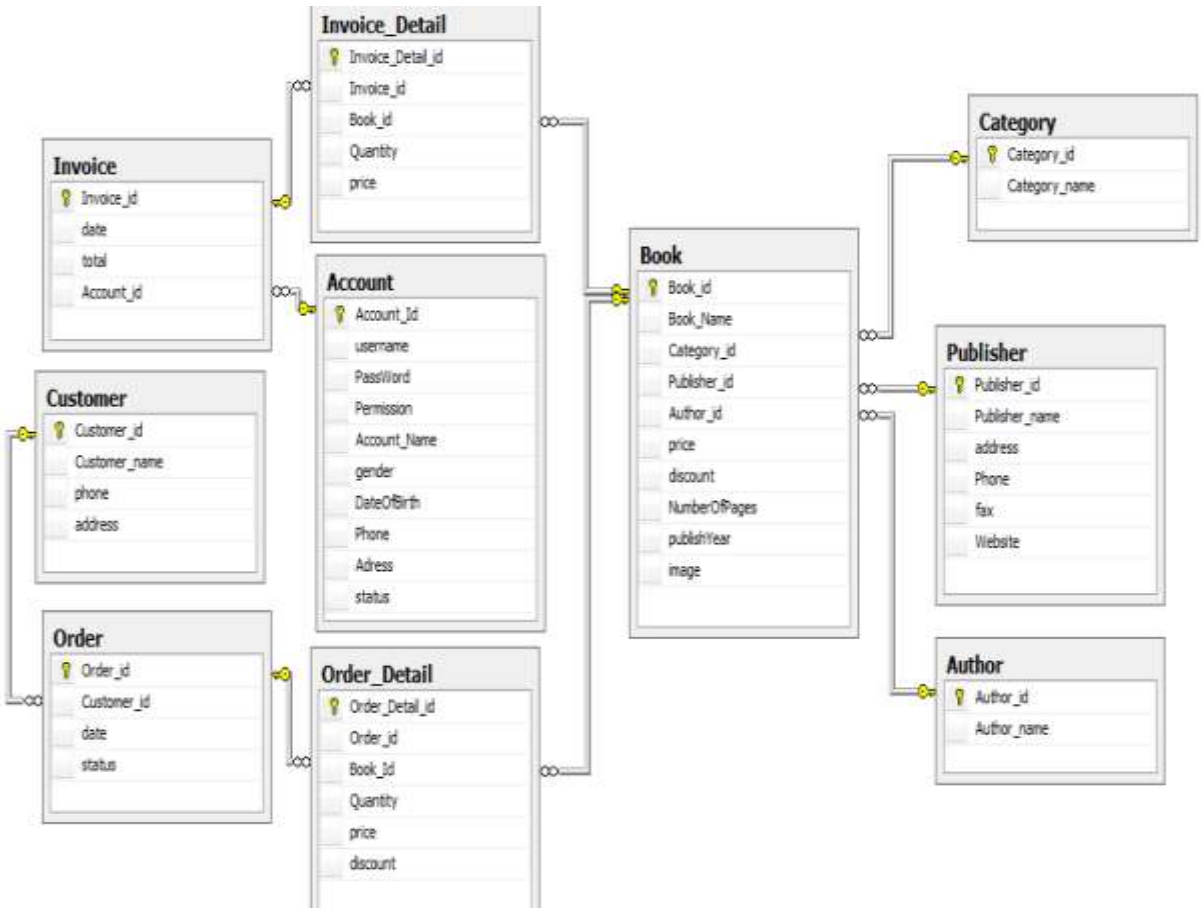
e.Báo cáo nhập sách :



f. Báo cáo mượn sách



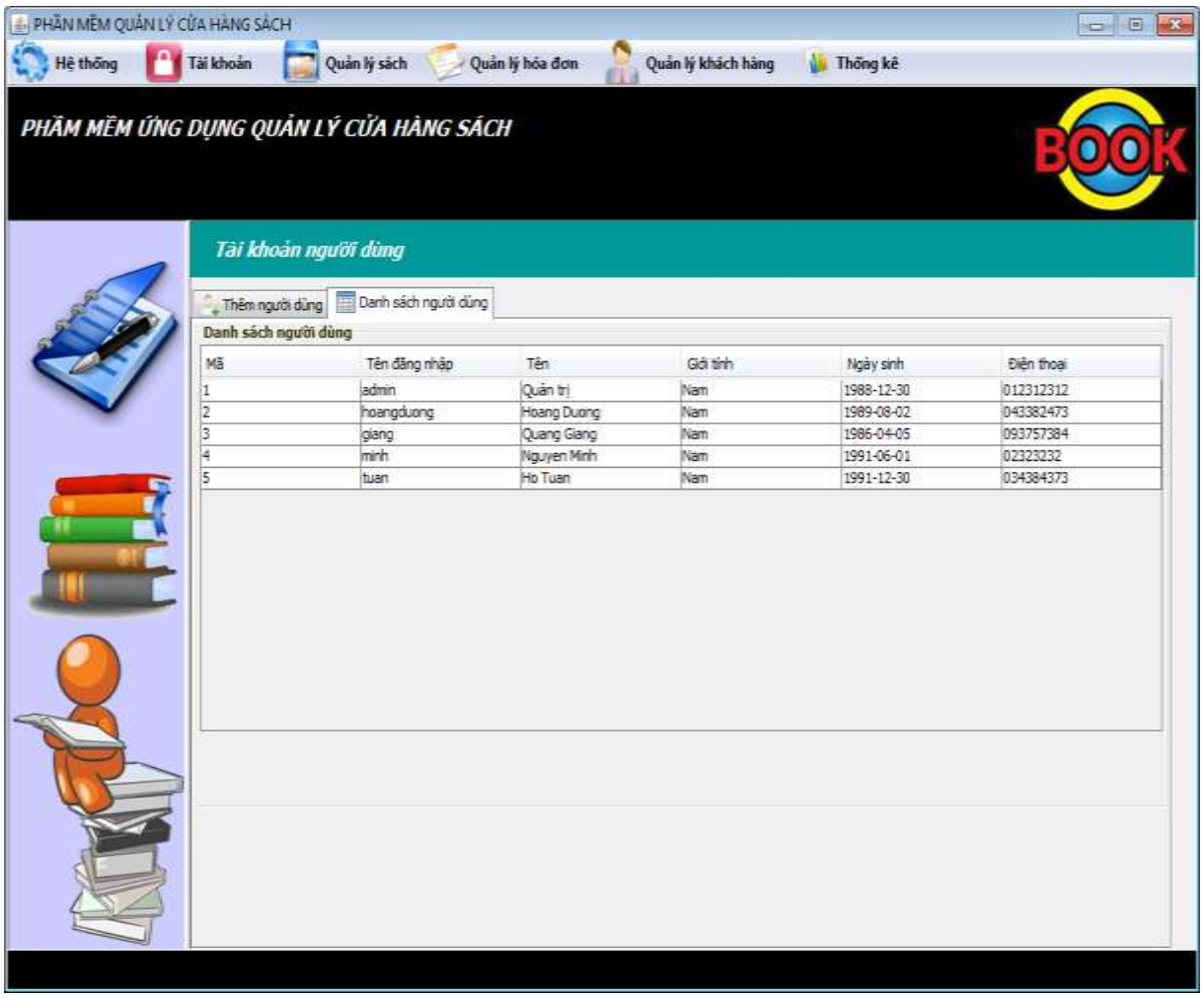
6.3. Thiết kế hệ thống



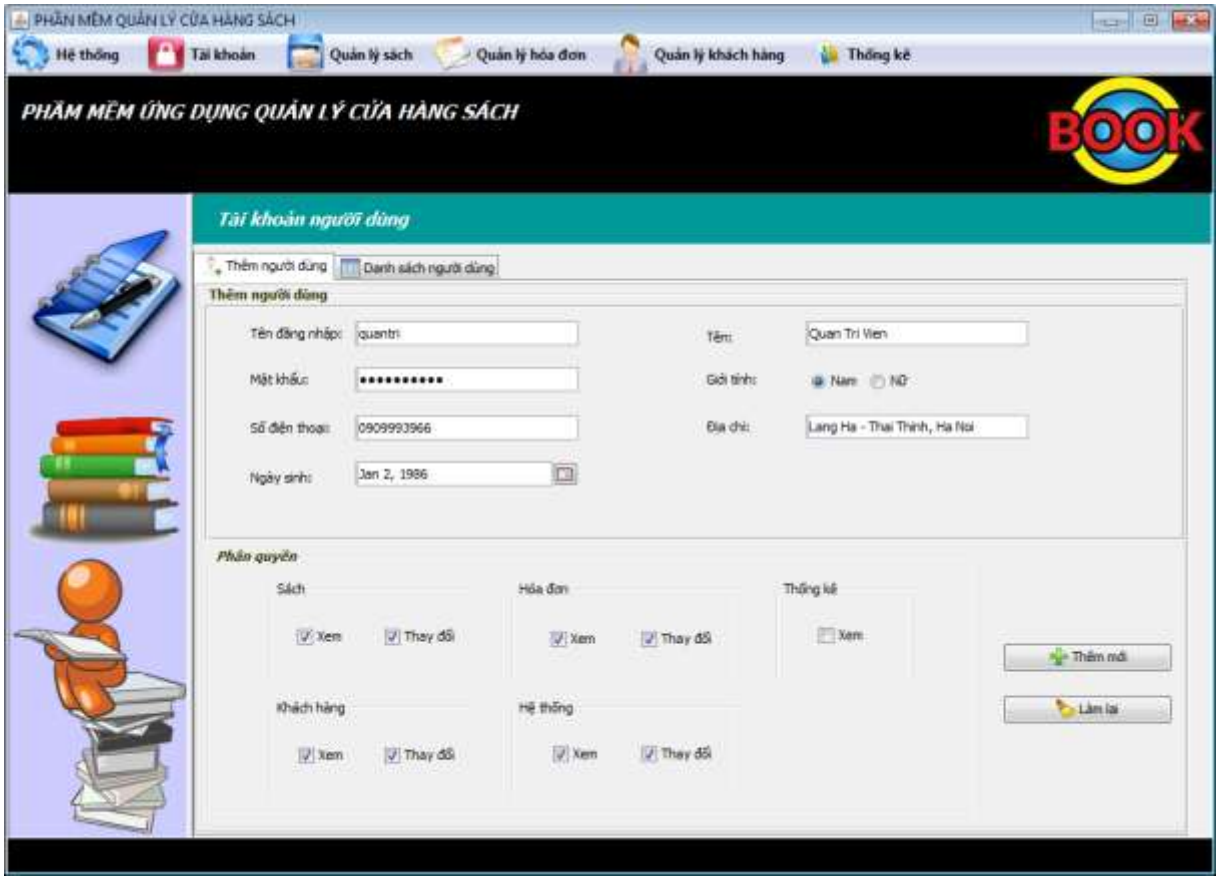
6.4. Một số giao diện chương trình

- Giao diện đăng nhập

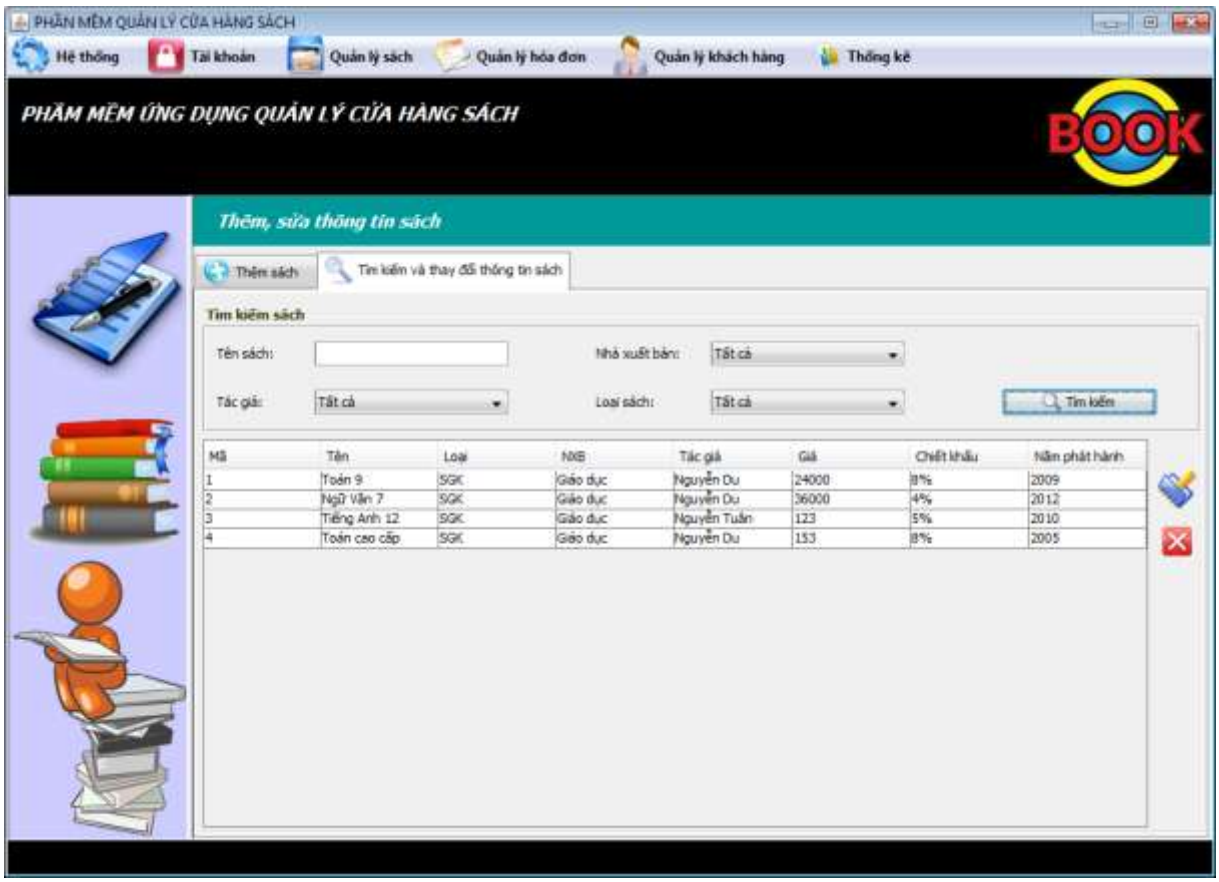
- Giao diện tài khoản người dùng



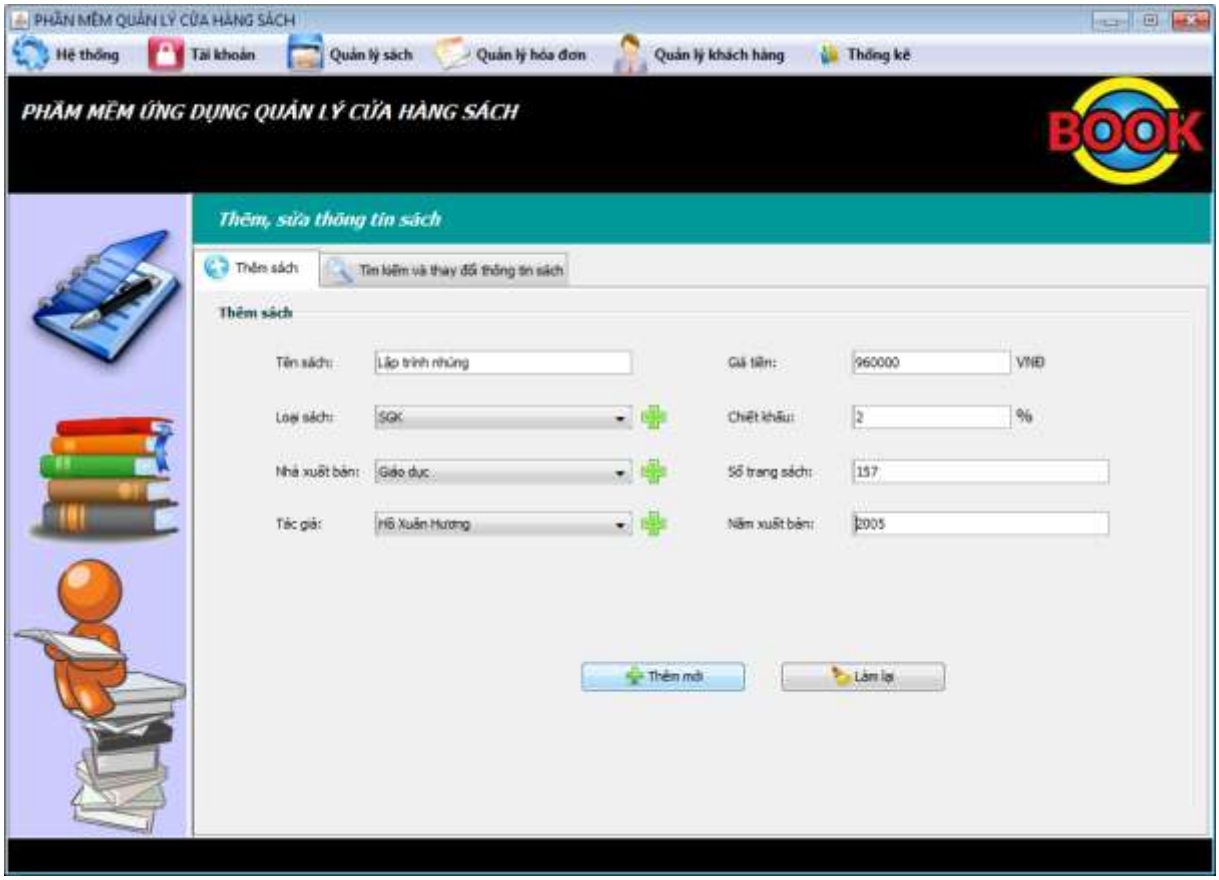
- **Giao diện thêm người dùng**



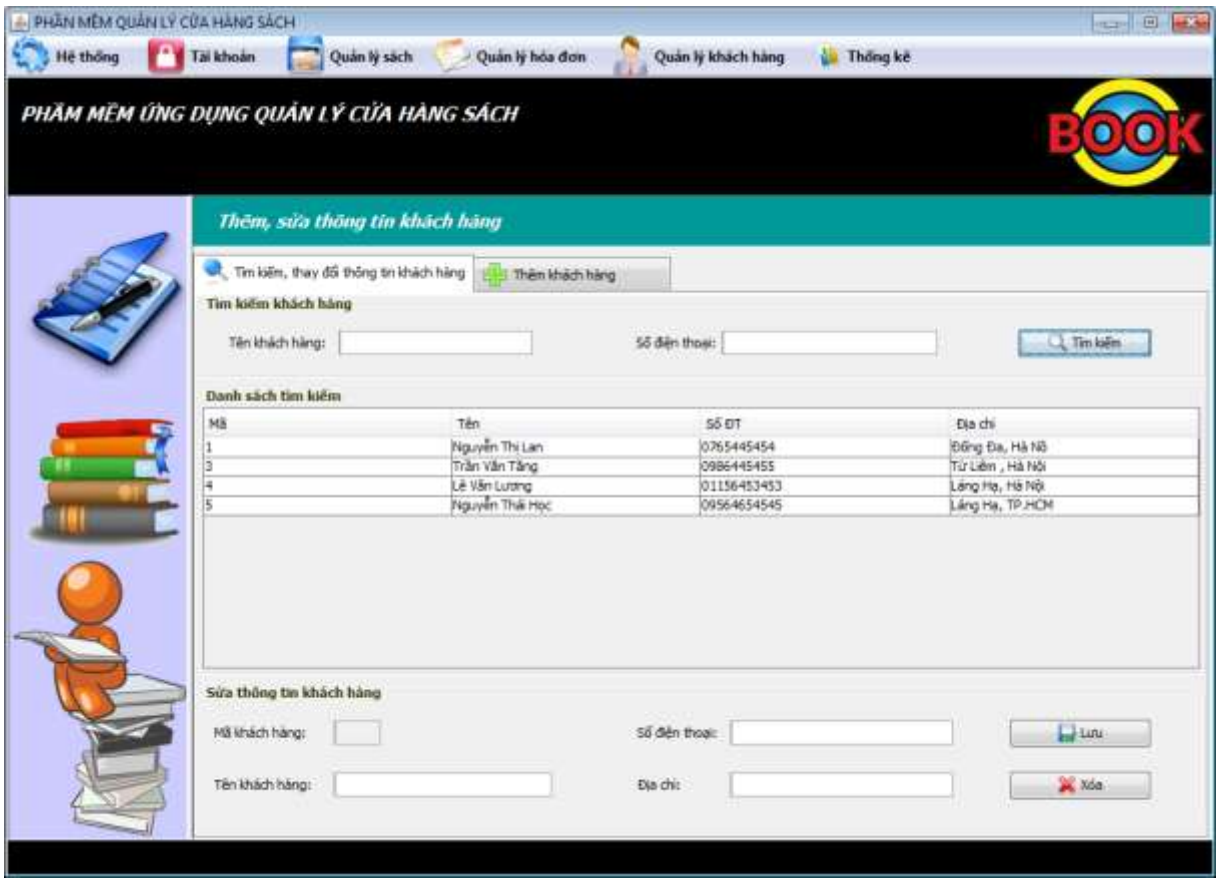
-Giao diện tìm kiếm sách



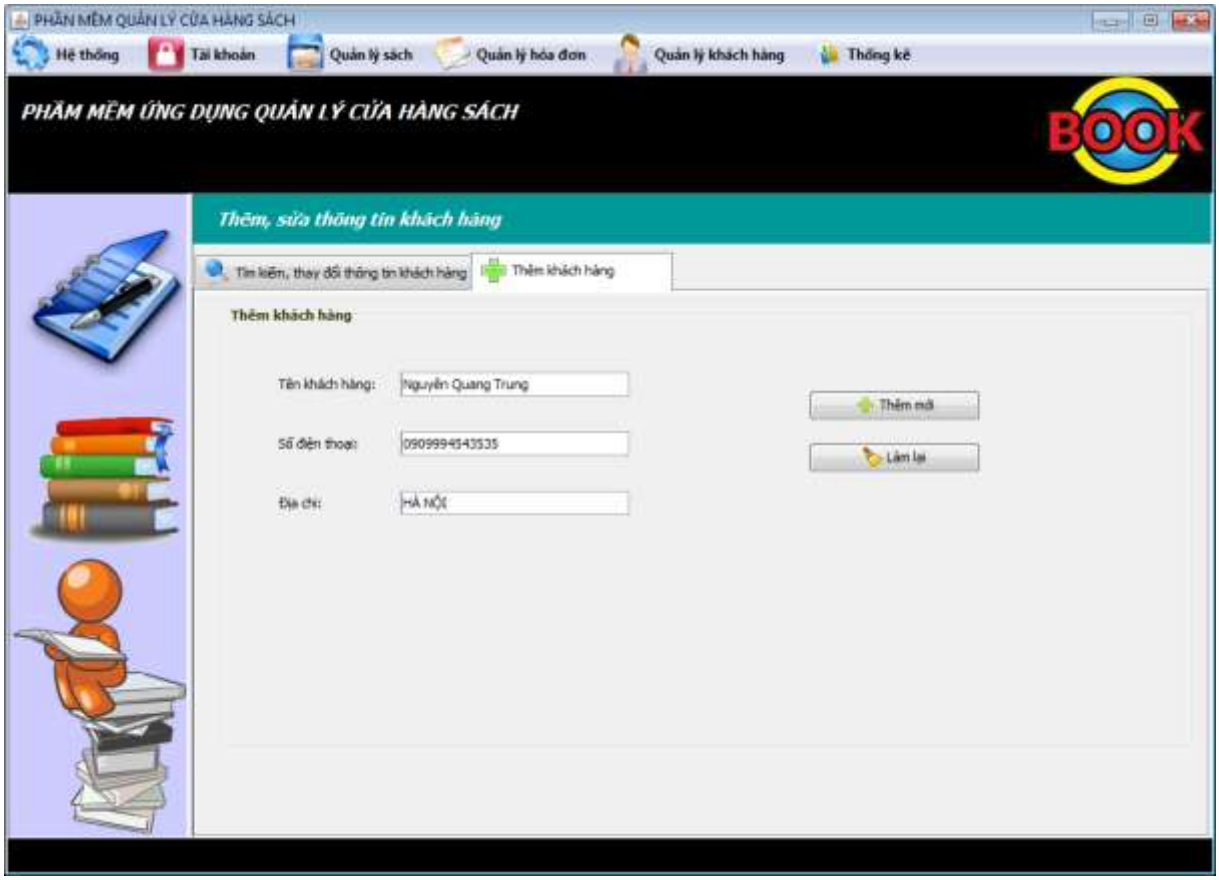
- Giao diện thêm sách



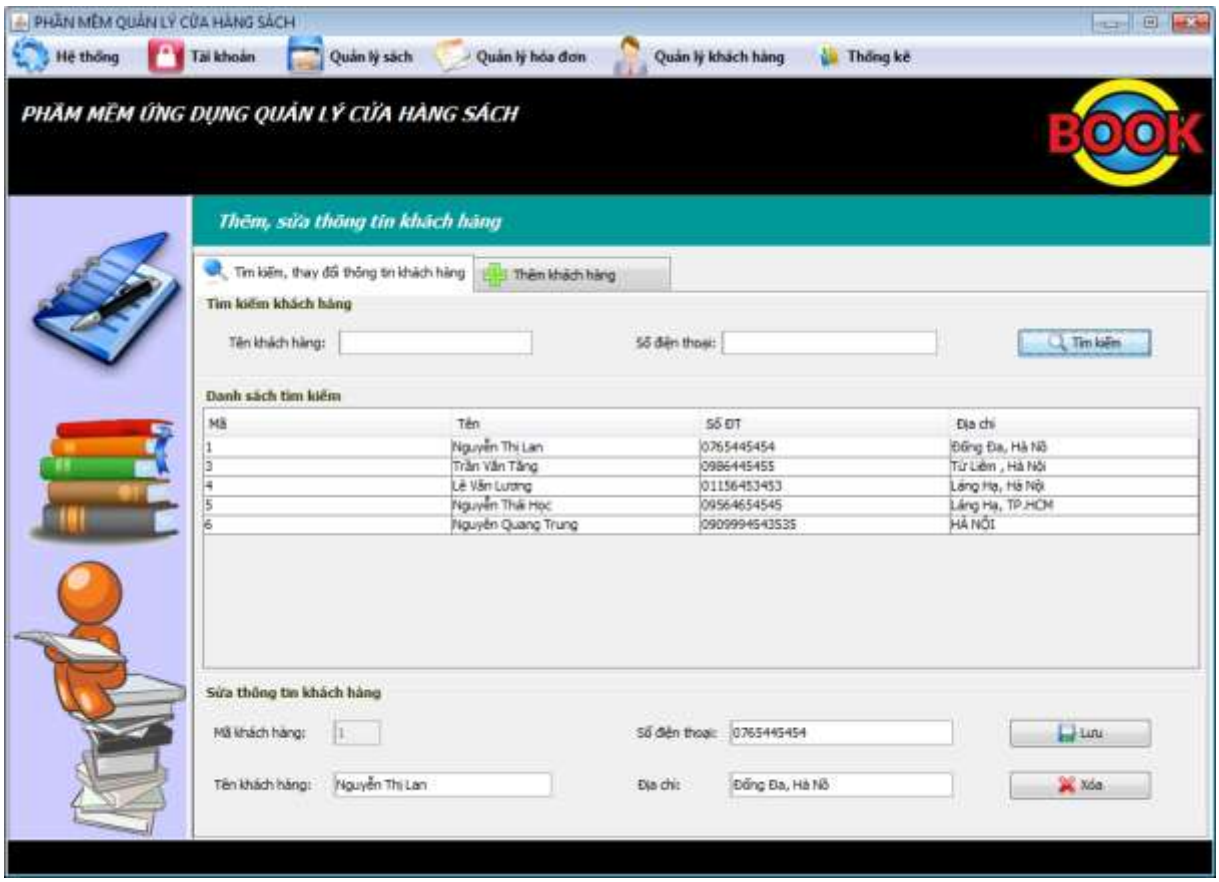
-Giao diện tìm kiếm khách hàng



- **Giao diện thêm khách hàng**



- Giao diện tìm kiếm, thay đổi thông tin khách hàng



-

- Giao diện thay đổi thông tin, thêm sách