

Chương 1: TỔNG QUAN VỀ LẬP TRÌNH JAVA (4:3)

1.1. Giới thiệu về ngôn ngữ lập trình Java

1.1.1. Java là gì?

Java là ngôn ngữ lập trình hướng đối tượng (tương tự C++) do Sun Microsystem đưa ra vào giữa thập niên 90.

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).

1.1.2. Lịch sử phát triển của ngôn ngữ lập trình Java

Ngôn ngữ lập trình Java do James Gosling và các công sự của Công ty Sun Microsystem phát triển.

Đầu thập niên 90, Sun Microsystem tập hợp các nhà nghiên cứu thành lập nên nhóm đặt tên là Green Team. Nhóm Green Team có trách nhiệm xây dựng công nghệ mới cho ngành điện tử tiêu dùng. Để giải quyết vấn đề này nhóm nghiên cứu phát triển đã xây dựng một ngôn ngữ lập trình mới đặt tên là Oak tương tự như C++ nhưng loại bỏ một số tính năng nguy hiểm của C++ và có khả năng chạy trên nhiều nền phần cứng khác nhau. Cùng lúc đó world wide web bắt đầu phát triển và Sun đã thấy được tiềm năng của ngôn ngữ Oak nên đã đầu tư cải tiến và phát triển. Sau đó không lâu ngôn ngữ mới với tên gọi là Java ra đời và được giới thiệu năm 1995.

Java là tên gọi của một hòn đảo ở Indonexia, Đây là nơi nhóm nghiên cứu phát triển đã chọn để đặt tên cho ngôn ngữ lập trình Java trong một chuyến đi tham quan và làm việc trên hòn đảo này. Hòn đảo Java này là nơi rất nổi tiếng với nhiều khu vườn trồng cafe, đó chính là lý do chúng ta thường thấy biểu tượng ly cafe trong nhiều sản phẩm phần mềm, công cụ lập trình Java của Sun cũng như một số hãng phần mềm khác đưa ra.

1.1.3. Một số đặc điểm nổi bật 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. Vấn đề này sẽ được bàn chi tiết trong chương 3.

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, ...

1.2. Các ứng dụng Java

1.2.1. 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");  
    }  
}
```

```
}  
}
```

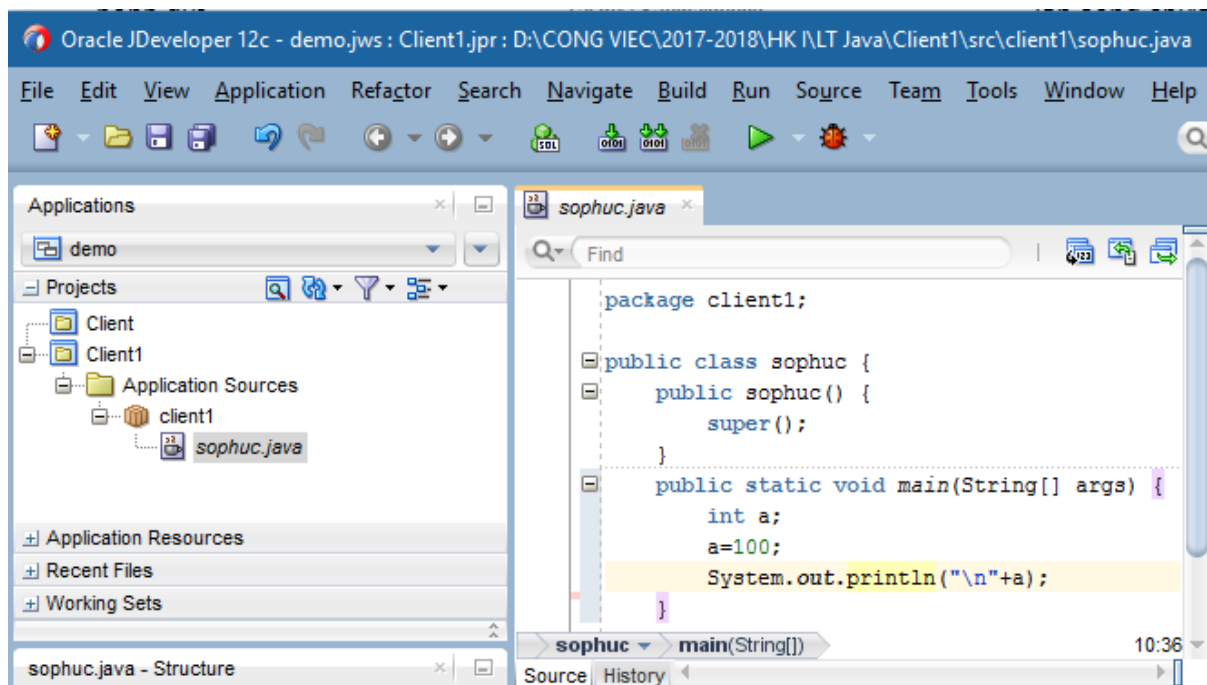
1.2.2. java và ứng dụng Applet

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 cạnh 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).

1.2.3. 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 chúng ta sẽ tìm hiểu chi tiết trong chương 4.

Minh họa thiết kế giao diện người dùng sử dụng JDeveloper



1.2.4. 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 Microsoft đang quảng cáo.

Hiện nay có rất nhiều trang Web nổi tiếng ở Việt Nam cũng như khắp nơi trên thế giới được xây dựng và phát triển dựa trên nền công nghệ Java. Số ứng dụng Web được xây dựng dùng công nghệ Java chắc chắn không ai có thể biết được con số chính xác là bao nhiêu, nhưng chúng tôi đưa ra đây vài ví dụ để thấy rằng công nghệ Java của Sun là một “đôi thủ đáng gờm” của Microsoft.

<http://java.sun.com/> <http://e-docs.bea.com/> <http://www.macromedia.com/software/jrun/>
<http://tomcat.apache.org/index.html>

Chắc không ít người trong chúng ta biết đến trang web thông tin nhà đất nổi tiếng ở TPHCM đó là: <http://www.nhadat.com/>. Ứng dụng Web này cũng được xây dựng dựa trên nền công nghệ java.

Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2EE tại địa chỉ: <http://java.sun.com/j2ee/>

1.2.5. 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.

Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2ME tại địa chỉ: <http://java.sun.com/j2me/>

1.3. Dịch và thực thi một chương trình viết bằng Java

Việc xây dựng, dịch và thực thi một chương trình viết bằng ngôn ngữ lập trình java có thể tóm tắt qua các bước sau:

- **Viết mã nguồn:** dùng một chương trình soạn thảo nào đấy (NotePad hay Jcreator chẳng hạn) để viết mã nguồn và lưu lại với tên có đuôi “.java”
- **Biên dịch ra mã máy ảo:** dùng trình biên dịch javac để biên dịch mã nguồn “.java” thành mã của máy ảo (java bytecode) có đuôi “. class” và lưu lên đĩa
- **Thông dịch và thực thi:** ứng dụng được load vào bộ nhớ, thông dịch và thực thi dùng trình thông dịch Java thông qua lệnh “java”.
- **Đưa mã java bytecode vào bộ nhớ:** đây là bước “loading”. Chương trình phải được đặt vào trong bộ nhớ trước khi thực thi. “Loader” sẽ lấy các files chứa mã java bytecode có đuôi “. class” và nạp chúng vào bộ nhớ.
- **Kiểm tra mã java bytecode:** trước khi trình thông dịch chuyển mã bytecode thành mã máy tương ứng để thực thi thì các mã bytecode phải được kiểm tra tính hợp lệ.
- **Thông dịch & thực thi:** cuối cùng dưới sự điều khiển của CPU và trình thông dịch tại mỗi thời điểm sẽ có một mã bytecode được chuyển sang mã máy và thực thi.

1.4. Thiết lập môi trường và xây dựng chương trình java cơ bản

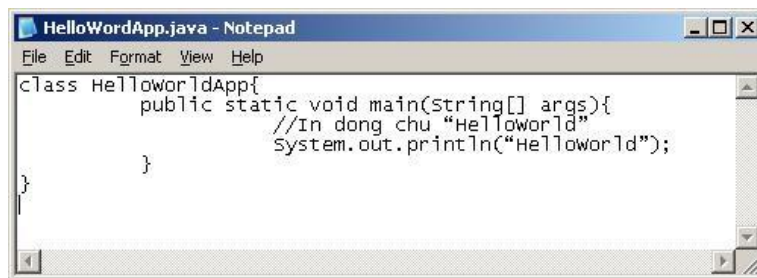
1.4.1. Tạo chương trình nguồn HelloWorldApp

- Khởi động chương trình soạn thảo (Notepad, Notepad++,...) và gõ đoạn mã sau

*/*Viết chương trình in dòng HelloWorld lên màn hình Console*/*

```
class HelloWorldApp{
    public static void main(String[] args){
        System.out.println("HelloWorld"); //In dòng chu HelloWorld
    }
}
```

Lưu lại với tên HelloWorldApp.java



1.4.2. Biên dịch tập tin nguồn HelloWorldApp

Việc biên dịch tập tin mã nguồn chương trình HelloWorldApp có thể thực hiện qua các bước cụ thể như sau:

- Mở cửa sổ Command Prompt.
- Chuyển đến thư mục chứa tập tin nguồn vừa tạo ra.
- Thực hiện câu lệnh: javac HelloWorldApp.java

Nếu gặp thông báo lỗi “Bad Command of filename” hoặc “The name specified is not recognized as an internal or external command, operable program or batch file” có nghĩa là Windows không tìm được trình biên dịch javac. Để sửa lỗi này chúng ta cần cập nhật lại đường dẫn PATH của hệ thống. Ngược lại nếu thành công bạn sẽ có thêm tập tin HelloWorldApp.class

1.4.3. Chạy chương trình HelloWorldApp

- Tại dấu nhắc gõ lệnh: java HelloWorldApp
- Nếu chương trình đúng bạn sẽ thấy dòng chữ HelloWorld trên màn hình Console.
- Nếu các bạn nhận được lỗi “Exception in thread "main" java.lang. NoClassDefFoundError: HelloWorldApp” có nghĩa là Java không thể tìm được tập tin mã bytecode tên HelloWorldApp.class của các bạn. Một trong những nơi java cố tìm tập tin bytecode là thư mục hiện tại của các bạn. Vì thế nếu tập tin byte code được đặt ở C:\java thì các bạn nên thay đổi đường dẫn tới đó.

1.4.4. Cấu trúc chương trình HelloWorldApp

Phương thức main(): là điểm bắt đầu thực thi một ứng dụng (chương trình chính). Mỗi ứng dụng Java phải chứa một phương thức main có dạng như sau:

public static void main(String[] args)

Phương thức main chứa ba chữ từ đặc tả sau:

- public: chỉ ra rằng phương thức main có thể được gọi bởi bất kỳ đối tượng nào.
- static: chỉ ra rằng phương thức main là một phương thức lớp.
- void: chỉ ra rằng phương thức main sẽ không trả về bất kỳ một giá trị nào.

Ngôn ngữ Java hỗ trợ ba kiểu chú thích sau:

/ text */*

// text

*/** documentation */*. Công cụ javadoc trong bộ JDK sử dụng chú thích này để chuẩn bị cho việc tự động phát sinh tài liệu.

- Dấu mở và đóng ngoặc nhọn “{” và “}”: là bắt đầu và kết thúc 1 khối lệnh.
- Dấu chấm phẩy “;” kết thúc 1 dòng lệnh.

1.4.5. Sử dụng phương thức/biến của lớp

Cú pháp: Tên_lớp.Tên_biến
 hoặc Tên_lớp.Tên_phương_thức(...)

Câu hỏi và bài tập

1. “Java là ngôn ngữ vừa biên dịch vừa thông dịch.” Hãy giải thích rõ điều này?
2. Hãy giải thích mối quan hệ giữa ngôn ngữ Java và C/C++ ?
3. Nêu khái quát quá trình xuất hiện ngôn ngữ lập trình Java?
4. Giải thích khái niệm về máy ảo Java, quá trình thực thi chương trình Java?
5. Download JDK tại địa chỉ: <http://java.sun.com/javase/downloads/index.jsp>. Hãy tiến hành cài đặt JDK và thiết lập biến môi trường.
6. Mở NotePad và soạn thảo chương trình in ra câu thông báo “Hello Java”. Lưu lại với tên là HelloJava.java; Mở Command Prompt, thực hiện biên dịch và thông dịch bằng lệnh: **javac [options] sourcecodename.java** và **java [options] classname**
7. Download và cài đặt công cụ lập trình java: hoặc JCreator hoặc JDeveloper hoặc NetBeans hoặc Eclipse hoặc bất kỳ công cụ nào anh (chị) thích.
8. Viết chương trình java in ra dòng chữ họ tên của anh (chị).

Chương 2: NGÔN NGỮ LẬP TRÌNH JAVA

2.1. Từ khóa, tên riêng, dòng ghi chú

2.1.1. Từ khóa

Keywords (Từ khóa) là những từ đã được định nghĩa cho trình biên dịch Java. Chúng có ý nghĩa đặc biệt cho trình biên dịch. Từ khóa Java phải có trong thông tin của bạn vì bạn không thể sử dụng chúng làm biến, lớp hoặc tên phương thức.

Một số từ khóa phổ biến:

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	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	true	false
null			

2.1.2. Tên riêng

Ngoài các từ khóa để thực hiện các chức năng nhất định, Java còn sử dụng một tập các tên riêng thường đặt cho các lớp hoặc phương thức.

Ví dụ: Math.PI

2.1.3. Ghi chú

Ghi chú hay còn gọi là chú thích là những dòng nhằm giải thích ý nghĩa cho một đoạn code nào đó để người lập trình dễ nhớ và rõ nghĩa. Trong quá trình dịch chương trình thì nội dung ghi chú sẽ không được dịch. Phần ghi chú là để người lập trình hoặc kiểm thử sử dụng.

Trong khi viết chương trình thì các dòng ghi chú vô cùng quan trọng vì có thể sau một thời gian chúng ta không còn làm việc với chương trình thì ý nghĩa hoặc thuật toán chúng ta có thể quên.

Nếu không có dòng ghi chú thì việc chỉnh sửa code sẽ gặp khó khăn đòi hỏi người lập trình phải xem và test code lại từ đầu.

Trong Java có 2 cách ghi chú thông dụng như sau:

Cách 1:

// Nội dung ghi chú

➔ Phần phía sau ký hiệu // là phần ghi chú. Với cách này thì nội dung ghi chú phải để trên 1 dòng

Cách 2:

/*

Dòng ghi chú 1

Dòng ghi chú 2

....

*/

➔ Phần ghi chú nằm giữa 2 ký hiệu /* */. Với cách này thì nội dung ghi chú có thể được ghi trên nhiều dòng. Nếu giữa 2 ký hiệu /* */ các bạn có viết code đi chăng nữa thì khi dịch chương trình Java cũng xem là ghi chú.

2.1.4. Lập trình Java trên NetBeans

2.1.4.1. Download và cài đặt IDE NetBeans

NetBeans là môi trường phát triển tích hợp (IDE) miễn phí hỗ trợ phát triển các dạng ứng dụng trên nhiều ngôn ngữ lập trình như: Java, C++, HTML, CSS, PHP,... Công cụ này có thể hoạt động tốt trên nhiều nền tảng như: Windows, Linux, MacOS,... Tuy nhiên IDE khá tốn dung lượng nên cần cấu hình máy khá ổn để chạy tốt công cụ này.

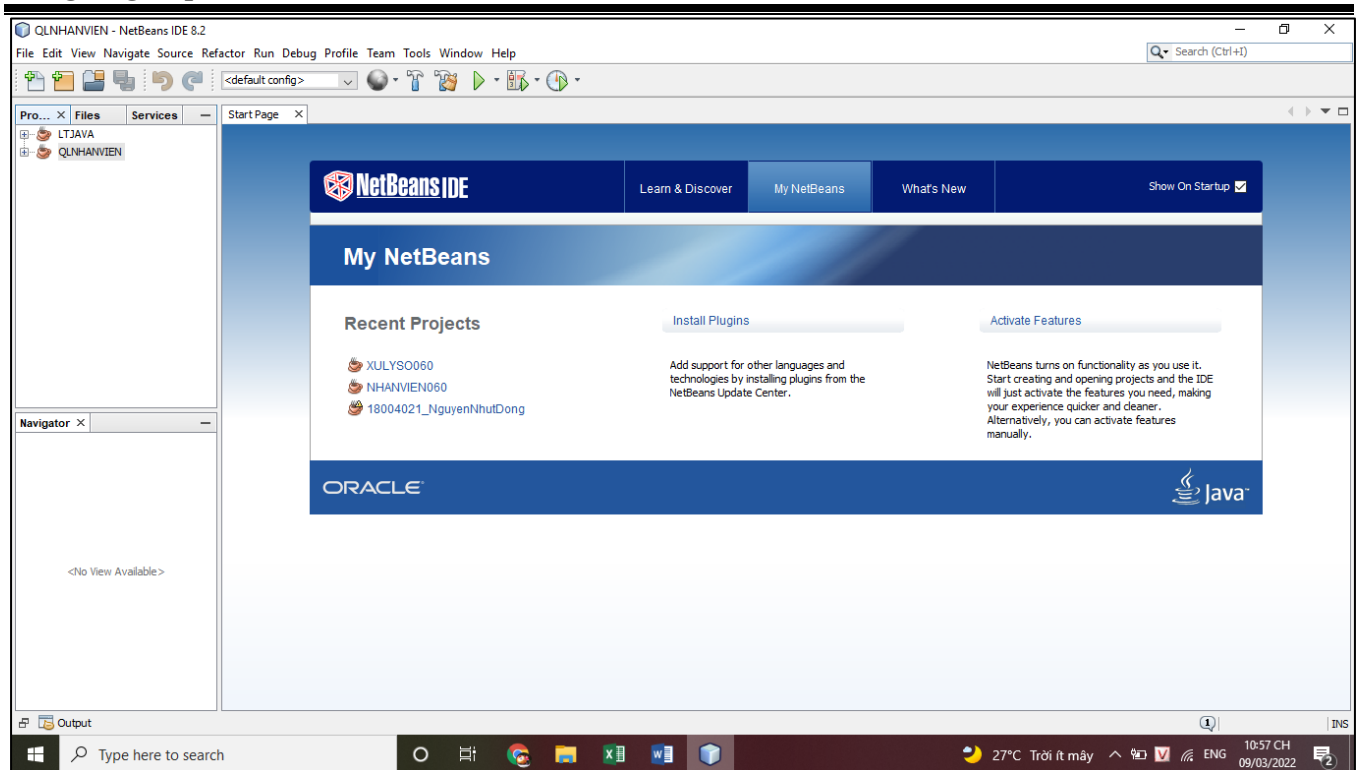
Trong số các tính năng khác là hệ thống dự án dựa trên Ant, hỗ trợ Maven, cải tiến mã nguồn, quản lý phiên bản (hỗ trợ CVS, Subversion, Git, Mercurial và Clearcase).

Nguồn download tại: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Lưu ý:

- Có nhiều phiên bản NetBeans, vì thế hãy lựa chọn phiên bản mới nhất hoặc phiên bản phù hợp với cấu hình của máy tính.
- Chọn gói cài đặt phù hợp với hệ điều hành.
- Chọn phiên bản có tích hợp JDK để không phải cài đặt thêm. Hoặc cài đặt JDK trước khi tiến hành cài đặt IDE NetBeans.
- Chọn nơi cài đặt phù hợp với dung lượng bộ nhớ

Sau khi cài đặt thành công NetBeans sẽ được giao diện như sau:



Giao diện khởi động NetBeans

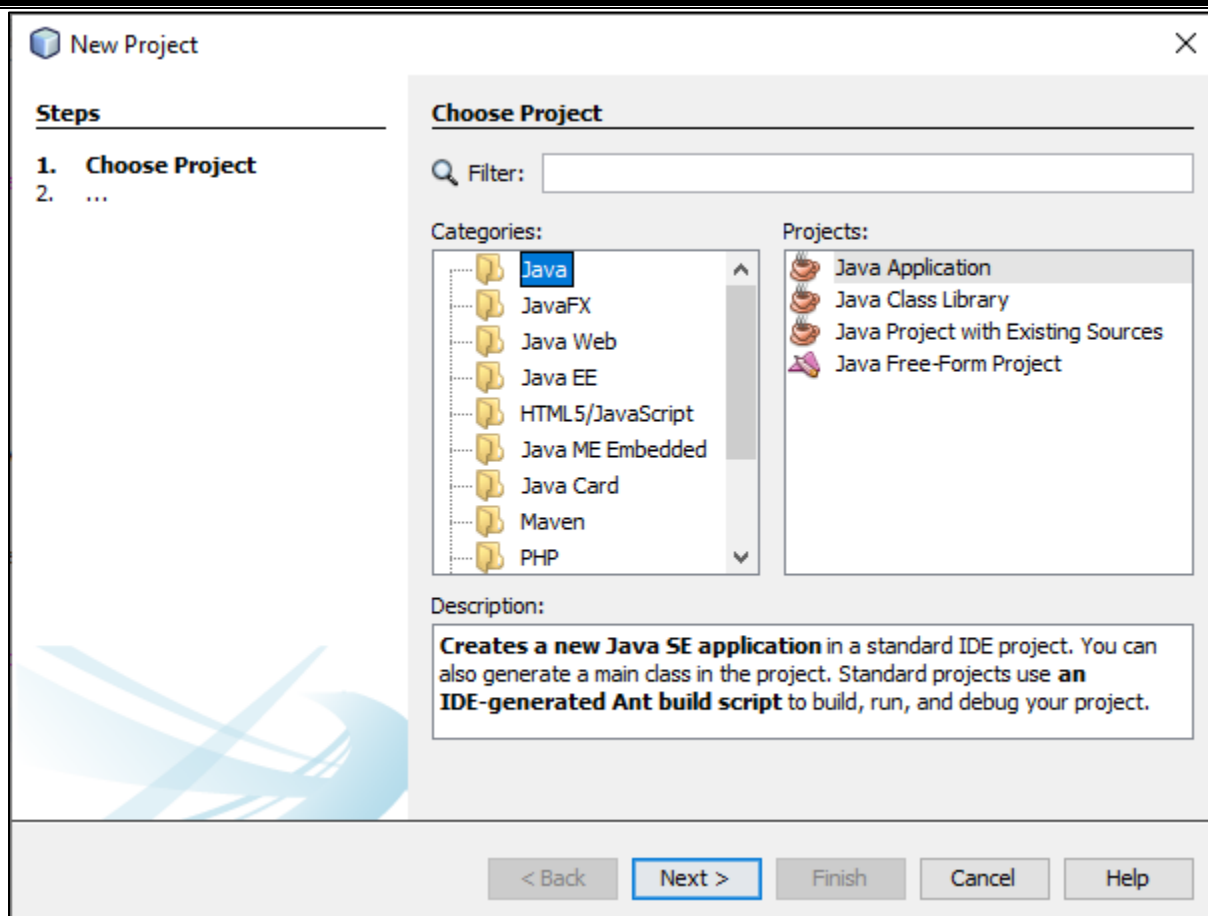
Tùy vào phiên bản cài đặt mà giao diện NetBeans sẽ khác nhau đôi chỗ.

2.1.4.2. Tạo project trong NetBeans

Project là dự án tập hợp nhiều module, các module này có thể sources, libraries, ..các module này thực chất là các công việc được chia nhỏ trong quá trình xây dựng phần mềm.

Các bước tạo project:

Bước 1: Từ Menu **File/ New Project** thì được giao diện như sau:



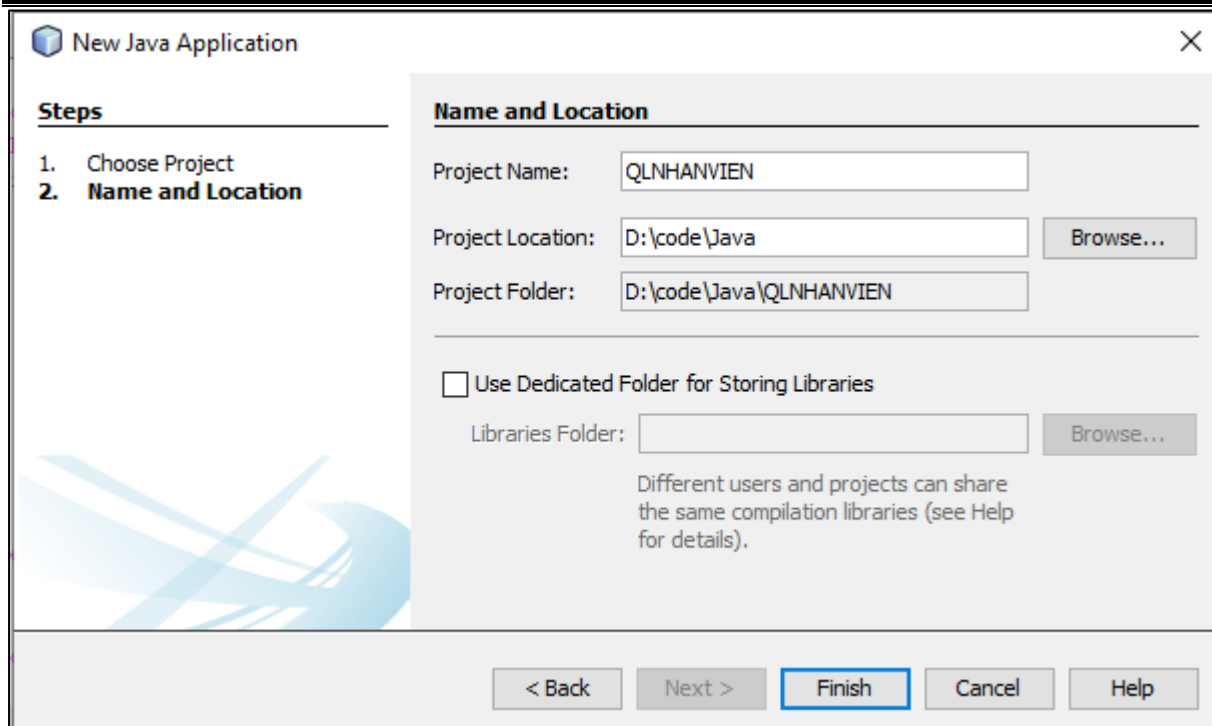
Giao diện tạo Project

Chọn các tiện ích phù hợp với nhu cầu của người lập trình ở danh mục Categories và loại Projects tương ứng.

Bước 2: Chọn gói ứng dụng cần sử dụng

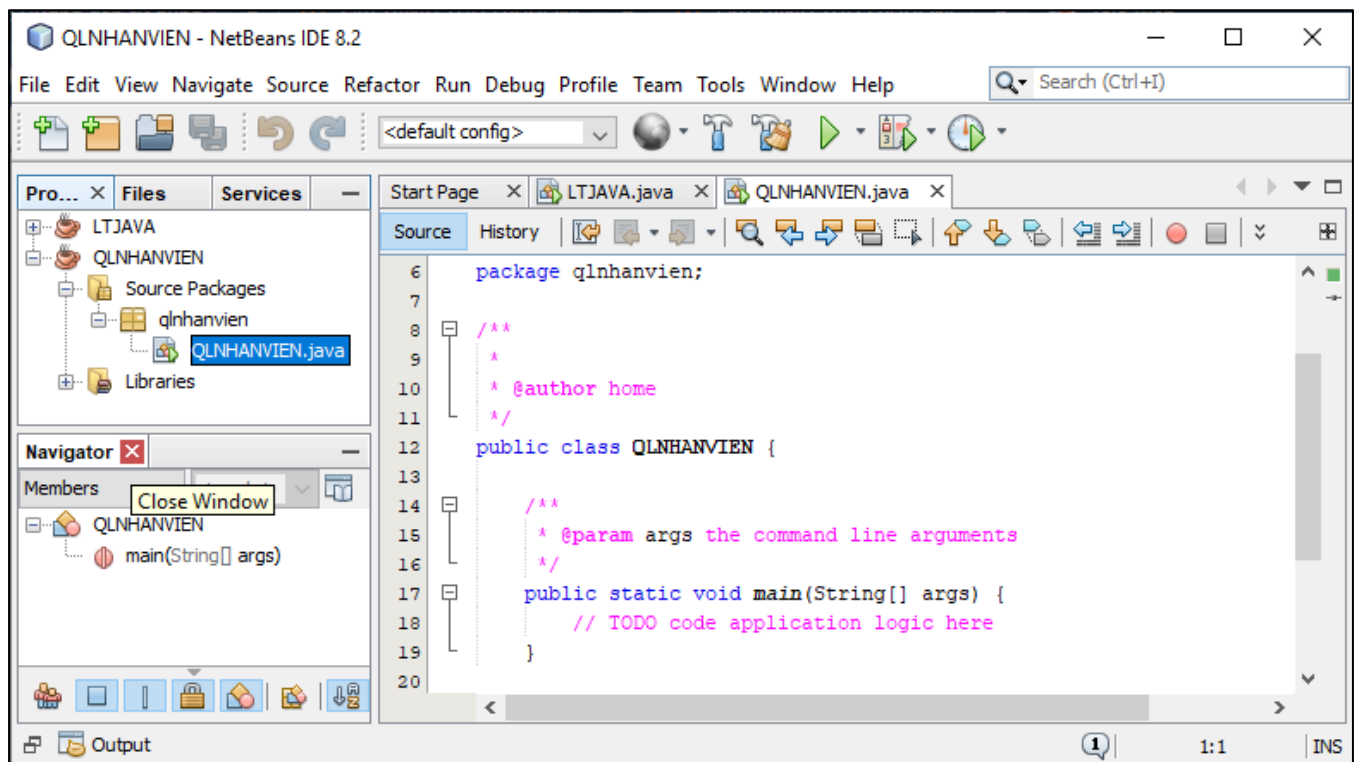
Nếu sử dụng NetBeans phục vụ cho lập trình ứng dụng thì chọn Categories là **Java** và Projects là **Java Application**

Bước 3: Đặt tên và chọn đường dẫn lưu trữ Projects



Đặt tên và chọn đường dẫn cho project

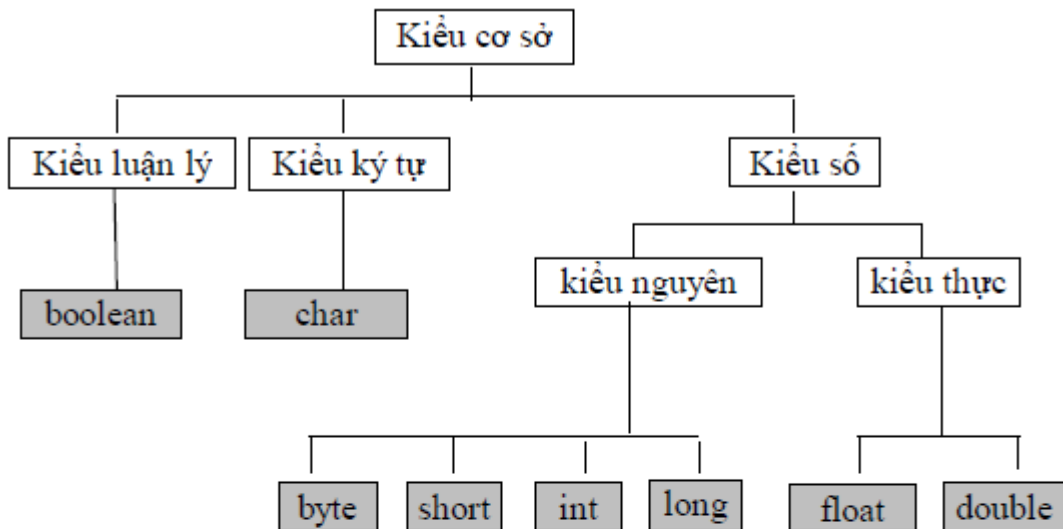
Khi tạo project thành công, NetBeans sẽ cung cấp một class có tên trùng với tên project vừa tạo



Với Project đặt tên là QLNHANVIEN thì NetBeans sẽ tạo class có tên QLNHANVIEN và cung cấp sẵn hàm main. Vì thế việc thực thi các code hoặc các module sẽ được triển khai bên trong hàm main của project. Mỗi project có 2 folder là Source Packages và Libraries.

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

Các kiểu dữ liệu cơ sở trong Java

2.2.1. Kiểu số nguyên

Java cung cấp 4 kiểu số nguyên khác nhau là: byte, short, int, long. Kích thước, giá trị nhỏ nhất, lớn nhất, cũng như giá trị mặc định của các kiểu dữ liệu số nguyên được mô tả chi tiết trong bảng trên. Kiểu mặc định của các số nguyên là kiểu int. Các số nguyên kiểu byte và short rất ít khi được dùng. Trong java không có kiểu số nguyên không dấu như trong ngôn ngữ C/C++.

Khai báo và khởi tạo giá trị cho các biến kiểu nguyên:

```
int x = 0;
```

```
long y = 100;
```


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 được chuyển sang kiểu int trước khi thực hiện.
- 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 chào");  
}
```

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.

 **Error(10,23): incomparable types: boolean and int**

2.2.2. 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 dấu 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.

2.2.3. 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.

2.2.4. 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. Biến và hằng

2.3.1. 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 ký tự chữ cái, một ký tự _ hay ký tự \$.

Tên biến không được trùng với các từ khóa (xem phụ lục các từ khóa trong java).

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

Cách khai báo biến

```
<kiểu_dữ_liệu> <tên_biến>;  
<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;
```

Gán giá trị cho biến

```
<tên_biến> = <giá_trị>;
```

Ví dụ: Khai báo số nguyên n, m và khởi tạo giá trị cho m và n.

```
int n;  
n=100;  
int m=150;
```

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 tượng dữ liệu cũng như các xử lý trong chương trình.

Ví dụ:

```
import java.lang.*; import java.io.*;  
class VariableDemo {  
    static int x, y;
```

```
public static void main(String[] args){
    x= 10;
    y= 20;
    int z = x+y;
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("z = x + y =" + z);
    System.out.println("So nho hon la so:" +
    Math.min(x, y));
    char c = 80;
    System.out.println("ky tu c la: " + c);
}
}
```

Kết quả chương trình

```
x = 10
y = 20
z = x + y =30
So nho hon la so:10
ky tu c la: P
Process exited with exit code 0.
```

2.3.2. 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 năm 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. Toán tử và biểu thức 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 nhọn ({}). 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 khối lệnh 3
```

2.4.1. Toán tử số học

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

++	Tăng 1
--	Giảm 1

2.4.2. Toán tử trên bit

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

2.4.3. 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
!	NOT (biểu thức logic)
	OR (biểu thức logic)
&&	AND (biểu thức logic)

2.4.4. 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;
```

```
int iCount = (int) fNum; // (iCount = 2)
```

2.4.5. 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.

2.4.6. Thứ tự ưu tiên

Thứ tự ưu tiên tính từ trái qua phải và từ trên xuống dưới

Cao nhất			
()	[]	.	
++	--	~	!
*	/	%	
+	-		
>>	>>> (dịch phải và điền 0 vào bit trống)	<<	
>	>=	<	<=
==	!=		
&			
^			
&&			
?:			
=	<toán tử>=		
Thấp nhất			

2.5. Nhập xuất từ bàn phím

2.5.1. Xuất dữ liệu

Lệnh xuất một **chuỗi** ra màn hình:

- Phương thức xuất và không xuống dòng

Cú pháp: System.out.print(StringValue);

Với StringValue và nội dung muốn xuất ra màn hình, StringValue có thể là hằng chuỗi, biến chuỗi hoặc biểu thức chuỗi.

Ví dụ: Xuất chuỗi “Hello!”

```
System.out.print(“Hello!”);
```

- Phương thức xuất chuỗi ra màn hình và xuống dòng

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

Ví dụ: Xuất chuỗi “Hello” và qua dòng mới

```
System.out.println(“Hello”);
```

2.5.2. Nhập dữ liệu

Đối tượng Scanner giúp chương trình nhận biết thông tin nhập từ bàn phím, Scanner có thể nhập đa dạng các kiểu dữ liệu với các phương thức riêng như nhập số nguyên có phương thức nextInt, Số thực có phương thức nextFloat,....

Các phương thức nhập liệu của Scanner dành riêng cho từng kiểu dữ liệu

- next(): Nhận vào từ đầu tiên các từ còn lại xem như không có.
- nextInt(): Nhận vào một số int
- nextLong(): Nhận vào một số long
- nextFloat(): Nhận vào một số float
- nextDouble(): Nhận vào một số double
- sc.nextLine(): Nhận vào một chuỗi String (Cả 1 câu)
- nextByte(): Nhận vào một byte
- nextBoolean(): Nhận vào một boolean

Cách sử dụng:

1. Import thư viện java.util.Scanner (Khai báo trước khai báo lớp, khai báo 1 lần sử dụng cho cả chương trình)
2. Khai báo đối tượng Scanner
3. Khai báo biến cần nhập
4. Sử dụng phương next tương ứng với kiểu dữ liệu cần nhập

Ví dụ 1: Nhập số nguyên

```
import java.util.Scanner;
```

```
public class Vd {
```

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

```
        System.out.print("Nhập số nguyên:");
```

```
        Scanner sc = new Scanner(System.in);
```

```
        int sn=sc.nextInt();
```

```
        System.out.print("số vừa nhập: "+sn);
```

```
    }
```

```
}
```

Ví dụ 2 nhập một chuỗi

```
import java.util.Scanner;

public class Vd2 {

    public static void main(String[] args) {
        System.out.print("Nhập một chuỗi:");
        Scanner sc = new Scanner(System.in);
        String st=sc.nextLine();
        System.out.print("Chuỗi vừa nhập: "+st);
    }
}
```

Lưu ý: trong trường hợp sử dụng Scanner áp dụng cho việc nhập một số rồi nhập 1 sẽ dẫn đến bị trôi lệnh (Java tự hiểu ký tự enter của số trước đó và gán cho câu lệnh nhập chuỗi, khi đó câu nhập chuỗi sẽ không hoạt động) trong trường hợp trên có thể sử dụng thêm phương thức `nextLine()` trước khi thực hiện nhập chuỗi hoặc sử dụng phương thức `nextLine()` để nhập dữ liệu cho tất cả các biến sau đó thực hiện ép kiểu phù hợp với biến.

Ví dụ Nhập tuổi và nhập họ tên của nhân viên

```
import java.util.Scanner;

public class NV {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nhập tuổi: ");
        int tuoi=sc.nextInt();
        sc.nextLine();
        System.out.print("Nhập họ tên: ");
        String hoten=sc.nextLine();
    }
}
```

Bên cạnh đó còn nhiều cách xử lý khác, anh (chị) có thể đề xuất thêm.

2.5.3. Ép kiểu trong java

Trong Java có 2 kiểu ép kiểu là nới rộng và thu hẹp:

1. Nới rộng (widening): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin.



2. Thu hẹp (narrowing): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin



Một số phương thức ép kiểu thông dụng:

Ép kiểu là quá trình chuyển giá trị của biểu thức thành một kiểu dữ liệu khác.

Chuyển chuỗi thành số nguyên: Integer.parseInt(String s)

Chuyển số nguyên thành chuỗi: Integer.toString(int n)

Chuyển chuỗi thành số thực: Double.parseDouble(String s)

Chuyển số thực thành chuỗi: Double.toString(double d)

2.6. Cấu trúc điều khiển trong java

2.6.1. Cấu trúc điều kiện if ... else

Dạng 1:

```
if (<điều_kiện>) {  
    <khởi_lệnh 1>;  
}
```

Dạng 2:

```
if (<điều_kiện>) {  
    <khởi_lệnh 1>;  
}  
else {  
    <khởi_lệnh 2>;  
}
```

Cấu trúc if thường được dùng để kiểm tra một điều kiện nào đó. Nếu điều kiện đúng thì thực hiện khối lệnh 1. Nếu có else thì thực hiện khối lệnh 2 ngược lại không thực hiện lệnh nào cả.

2.6.2. Cấu trúc switch ... case

```
switch (<biến>) {  
    case <giá trị_1>:    <khởi_lệnh_1>;  
                        break;  
    ....  
    case <giá trị_n>:    <khởi_lệnh_n>;  
                        break;
```

default: <khởi_lệnh_default>;

}

Cấu trúc switch thường dùng để lựa chọn 1 công việc thực hiện trong danh sách công việc có sẵn tùy thuộc vào từng giá trị cụ thể của biến.

Trong trường hợp giá trị <biến> trùng với giá trị tại 1 case nào đó mà case không có câu lệnh break thì công việc ở các case phía sau sẽ tiếp tục thực hiện cho đến khi gặp break hoặc thực hiện hết các case phía sau.

2.6.3. Cấu trúc lặp

Dạng 1: while (...)

```
while (điều_kiện_lặp) {  
    <khởi_lệnh>;  
}
```

Dạng 2: do { ... } while;

```
do {  
    <khởi_lệnh>;  
} while (<điều_kiện>);
```

Cấu trúc while và cấu trúc do... while thường dùng để lặp lại 1 công việc dựa vào điều kiện. Quá trình lặp sẽ được thực hiện khi điều_kiện_lặp còn đúng và sẽ dừng lại khi điều_kiện_lặp bị sai. Vì thế trong quá trình thiết kế code cho công việc (khởi_lệnh) trong while phải có ít nhất 1 dòng lệnh làm thay đổi giá trị của điều_kiện_lặp hoặc câu lệnh thoát khỏi vòng lặp (có thể là break hoặc exit). Việc sử dụng cấu trúc nào tùy thuộc vào mục đích và thói quen của người lập trình.

Dạng 3: for (...)

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

Bản chất của cấu trúc for là lặp lại công việc với số lần biết trước nhưng cũng có sử dụng tương tự như while hoặc do... while.

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 đến 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 này 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 từ 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 những 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. Mảng 2 chiều

Khai báo mảng 2 chiều trong java

<Kiểu dữ liệu> [][] <Tên mảng>;

hoặc

<Kiểu dữ liệu> <Tên mảng> [][]

Ví dụ:

```
int a[][];
```

```
int[][] a;
```

Khởi tạo mảng đa chiều:

<Tên mảng> [][] = new <Kiểu dữ liệu> [<Số dòng>][<Số cột>]

Ví dụ:

```
int a[][] = new int[2][3];
```

Cấp phát phần tử cho mảng đa chiều

```
int a[][] = {  
    {3, 4},  
    {2, 8},  
};
```

hoặc

```
a[0][0] = 3;
```

```
a[0][1] = 4;
```

```
a[1][0] = 2;
```

```
a[1][1] = 8;
```

2.8. Xâu ký tự, các hàm toán học và hàm thời gian

2.8.1. Xâu ký tự

Trong Java, để khai báo 1 chuỗi ký tự thì chúng ta có 2 cách như sau:

Cách 1:

Cú pháp: String TenChuoi = "Giá trị khởi tạo";

Ví dụ: Khai báo chuỗi quốc tịch gán sẵn Việt Nam cho biến

```
String QuocTich = "Việt Nam";
```

Cách 2: st2

Cú pháp:

```
String TenChuoi = new String("Giá trị khởi tạo");
```

Ví dụ: Khai báo chuỗi quốc tịch gán sẵn Việt Nam cho biến

```
String QuocTich = new String("Việt Nam");
```

2.8.2. Các phương thức xử lý xâu ký tự

length(): Xác định chiều dài của chuỗi

concat(str1, str2): Trả về chuỗi mới sau khi ghép chuỗi str1 với chuỗi str2

charAt(int index): Trả về ký tự trong chuỗi tại vị trí index

compareTo(Str2): So sánh với chuỗi Str2 trả về 1 trong 3 giá trị. Nếu là 0 là bằng với chuỗi Str2, nếu là 1 là nhỏ hơn chuỗi Str2, ngược lại là lớn hơn chuỗi Str2.

indexOf(Str2): Trả về vị trí xuất hiện đầu tiên của chuỗi Str2 trong chuỗi, Nếu kết quả trả về <0 có nghĩa là chuỗi Str2 không có trong chuỗi.

indexOf(String string2): Trả về vị trí xuất hiện cuối của chuỗi Str2 trong chuỗi, Nếu kết quả trả về <0 có nghĩa là chuỗi Str2 không có trong chuỗi.

replace(StrOld, StrNew): Trả về chuỗi mới sau khi thay thế chuỗi cũ StrOld bằng chuỗi mới StrNew

trim(): Trả về chuỗi mới sau khi xóa tất cả khoảng trắng thừa ở đầu và cuối chuỗi.

Lưu ý: Các phương thức trên muốn sử dụng được phải có đối tượng chuỗi tham chiếu tới thông qua toán tử chấm (.)

Ví dụ: thay chuỗi “trang” bằng chuỗi “Trung” trong chuỗi hoten

```
hoten=hoten.replace(“trang”,”Trung”);
```

Ví dụ: Tìm vị trí chuỗi “Lan” trong chuỗi ht

```
int vt=ht.indexOf(“Lan”);
```

2.8.3. Các hàm toán học

Các hàm toán học trong java thuộc thư viện Math vì thế cần khai báo thư viện Math trước khi sử dụng để thuận tiện cho việc sử dụng trong chương trình.

Math.PI: Trả về giá trị của pi.

Math.abs(S): Trả về trị tuyệt đối của số S.

Math.ceil(S): Trả về số nguyên tăng gần S nhất.

Math.floor(S): Trả về số nguyên giảm gần S nhất.

Math.max(S1,S2): Trả về số lớn hơn trong hai số S1 và S2.

Math.min(S1,S2): Trả về số nhỏ hơn trong hai số S1 và S2.

Math.pow(x,y): Trả về giá trị của x lũy thừa y.

Math.sqrt(S): Trả về căn bậc hai của số S.

Ví dụ: Tính 2 lũy thừa 3

```
float kq=Math.pow(2,3);
```

2.8.4. Các hàm thời gian

- Khởi tạo ngày:

`Date.valueOf(String StrDate);`

Ví dụ: Khởi tạo đối tượng date là ngày 15/03/2022

`Date date = Date.valueOf("2022-03-15");`

Việc định dạng ngày tháng năm như thế nào tùy thuộc vào phương thức `SimpleDateFormat(String StrFormatDate)`

- Lấy ngày tháng năm hiện tại của hệ thống

`Calendar cal = Calendar.getInstance();`

`Date date = cal.getTime();`

Việc truy xuất thông tin của đối tượng *date* có các phương thức hỗ trợ sau:

`.getDay()`: trả về ngày trong tháng của date

`.getMonth()`: trả về tháng trong năm của date

`.getYear()`: trả về năm của date

`.getHours()`: trả về giờ của date

`.getMinutes()`: trả về phút của date

`.getSeconds()`: trả về giây của date

Lưu ý:

- Khi sử dụng đối tượng của lớp `Calendar` cần import thư viện `java.util.Calendar`;
- Khi sử dụng đối tượng của lớp `Date` cần import thư viện `java.util.Date`;
- Việc import thư viện sẽ được thực hiện ở vị trí ngoài khai báo lớp như hình bên dưới.

```
8  import java.util.Calendar;
9  import java.util.Date;
10
11  public class LTJAVA {
12
13      /**
14       * @param args the command line arguments
15       */
16      public static void main(String[] args) {
17          // TODO code application logic here
18      }
19  }
```

2.9- Xử lý ngoại lệ - Exception

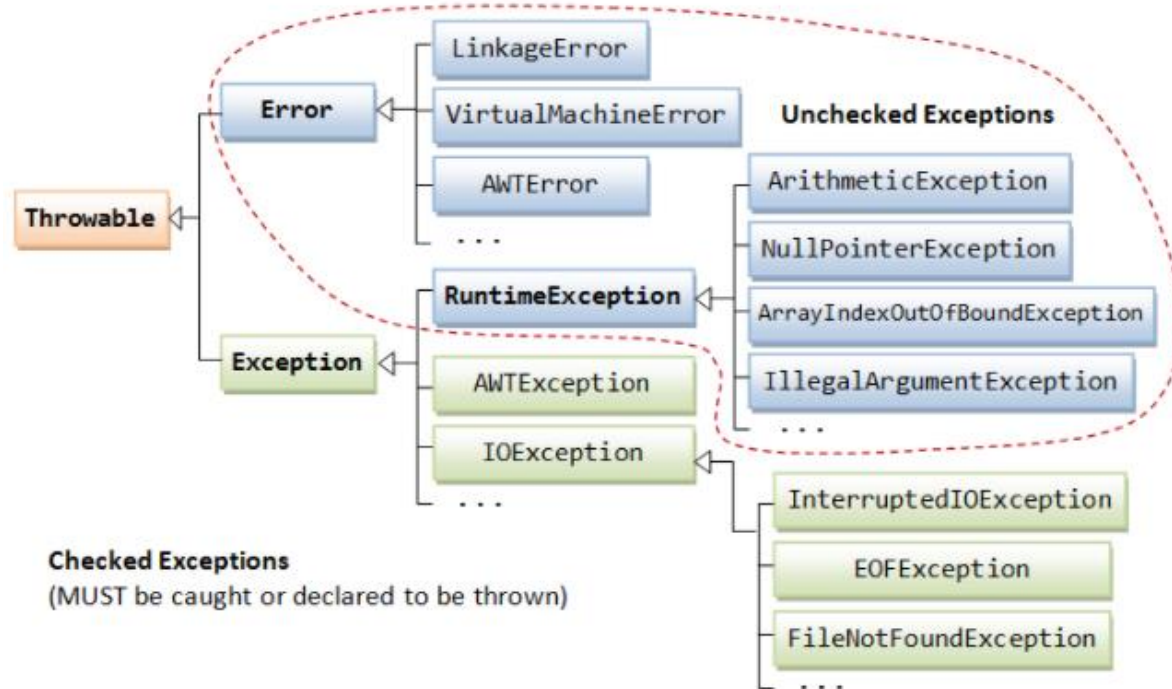
Exception là một sự kiện xảy ra trong quá trình thực thi một chương trình Java, nó làm phá vỡ cái flow (luồng xử lý) bình thường của một chương trình, thậm chí chết chương trình.

Một ngoại lệ có thể xảy ra với nhiều lý do khác nhau, nó nằm ngoài dự tính của chương trình. Một vài ngoại lệ xảy ra bởi lỗi của người dùng, một số khác bởi lỗi của lập trình viên và số khác nữa đến từ lỗi của nguồn dữ liệu vật lý. Chẳng hạn như:

- Người dùng nhập dữ liệu không hợp lệ.
- Truy cập ngoài chỉ số mảng.

- Một file cần được mở nhưng không thể tìm thấy.
- Kết nối mạng bị ngắt trong quá trình thực hiện giao tiếp hoặc JVM hết bộ nhớ.
-

2.9.1. Hệ thống cấp bậc của các lớp ngoại lệ trong Java



Sơ đồ phân cấp Exception trong java

Đây là mô hình sơ đồ phân cấp của Exception trong java.

- Class ở mức cao nhất là Throwable
- Hai class con trực tiếp là Error và Exception.

Trong nhánh **Exception** có một nhánh con **RuntimeException** là các ngoại lệ sẽ không được java kiểm tra trong thời điểm biên dịch.

2.9.2. Exception

Trong Java có 2 loại exception: checked và unchecked. Tất cả các checked exception được kế thừa từ lớp Exception ngoại trừ lớp RuntimeException. RuntimeException là lớp cơ sở của tất cả các lớp unchecked exception. Đó cũng là dấu hiệu để nhận biết đâu là checked exception và đâu là unchecked exception.

Điểm khác biệt giữa các lớp checked và unchecked exception chính là thời điểm xác định được exception có thể xảy ra.

2.9.2.1. Checked exceptions

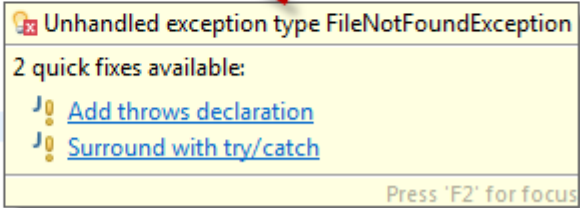
Là loại exception xảy ra trong lúc compile time, nó cũng có thể được gọi là **compile time exceptions**. Loại exception này không thể bỏ qua được trong quá trình compile, bắt buộc ta phải handle nó.

Các lớp extends từ lớp Throwable ngoại trừ RuntimeException và Error được gọi là checked exception.

Ví dụ: IOException, FileNotFoundException, NoSuchFieldException,

Ví dụ chương trình sau đọc file sử dụng java.io.FileReader lớp này ném ra ngoại lệ FileNotFoundException. Trình biên dịch thông báo lỗi như sau:

```
3 import java.io.FileReader;
4
5 public class ExceptionExample2 {
6
7     public static void main(String[] args) {
8
9         FileReader f = new FileReader("File is not exists");
10    }
11 }
12
13
14
15
16
17
18
19
```



```
45 public class FileReader extends InputStreamReader {
46
47     /**
48      * Creates a new <tt>FileReader</tt>, given the name of the
49      * file to read from.
50      *
51      * @param fileName the name of the file to read from
52      * @exception FileNotFoundException if the named file does not exist,
53      *         is a directory rather than a regular file,
54      *         or for some other reason cannot be opened for
55      *         reading.
56      */
57     public FileReader(String fileName) throws FileNotFoundException {
58         super(new FileInputStream(fileName));
59     }
60 }
```

2.9.2.2. UnChecked exceptions

Là loại exception xảy ra tại thời điểm thực thi chương trình, nó cũng có thể gọi là **runtime exceptions** đó là programming bugs, lỗi logic của chương trình... Loại exception này được bỏ qua trong quá trình compile, không bắt buộc ta phải handle nó.

Các lớp extends từ RuntimeException được gọi là unchecked exception.

Ví dụ: NullPointerException, NumberFormatException, ArrayIndexOutOfBoundsException, DivideByZeroException, ...

Ví dụ một biến có giá trị null, thực hiện bất kỳ hoạt động nào bởi biến đó sẽ xảy ra ngoại lệ NullPointerException.

```
3 public class ExceptionExample3 {  
4  
5     public static void main(String[] args) {  
6  
7         String obj = null;  
8         System.out.println(obj.length()); // NullPointerException  
9     }  
10  
11 }
```

```
Exception in thread "main" java.lang.NullPointerException  
    at com.gpcoder.exception.ExceptionExample3.main(ExceptionExample3.java:10)
```

2.9.3. Error

Error là những vấn đề nghiêm trọng liên quan đến môi trường thực thi của ứng dụng hoặc hệ thống mà lập trình viên không thể kiểm soát. Nó thường làm chết chương trình.

Lớp Error định nghĩa các ngoại lệ mà không thể bắt (catch) từ chương trình.

Ví dụ: OutOfMemoryError, VirtualMachineError, and StackOverflowError, ...

Ví dụ chương trình đệ quy vô tận:

```
3 public class ExceptionExample4 {  
4     public static void print() {  
5         print();  
6     }  
7  
8     public static void main(String[] args) {  
9         print();  
10    }  
11  
12 }
```

```
Exception in thread "main" java.lang.StackOverflowError  
    at com.gpcoder.exception.ExceptionExample4.print(ExceptionExample4.java:6)  
    at com.gpcoder.exception.ExceptionExample4.print(ExceptionExample4.java:6)  
    at com.gpcoder.exception.ExceptionExample4.print(ExceptionExample4.java:6)  
    at com.gpcoder.exception.ExceptionExample4.print(ExceptionExample4.java:6)  
    at com.gpcoder.exception.ExceptionExample4.print(ExceptionExample4.java:6)
```

2.9.4. Xử lý ngoại lệ trong Java

Khối lệnh try trong java được sử dụng để chứa một đoạn code có thể xảy ra một ngoại lệ. Nó phải được khai báo trong phương thức.

Sau một khối lệnh try bạn phải khai báo khối lệnh catch hoặc finally hoặc cả hai.

2.9.4.1. Cú pháp của khối lệnh try-catch

```
try{
    // code có thể xuất hiện ngoại lệ

} catch(Exception_class_Name ex){
    // code xử lý khi gặp ngoại lệ
}
```

2.9.4.2. Cú pháp của khối lệnh try-catch-finally

```
try {
    // code có thể xuất hiện ngoại lệ
} catch(Exception_class_Name_1 ex) {
    // code xử lý ngoại lệ 1
} catch(Exception_class_Name_2 ex) {
    // code xử lý ngoại lệ 2
} catch(Exception_class_Name_n ex) {
    // code xử lý ngoại lệ n
} finally {
    // code trong khối này luôn được thực thi
}
```

2.9.4.3. Ví dụ xử lý ngoại lệ

Chuyển một chuỗi thành số nguyên. Ví dụ chuyển chuỗi "123a34" thành số nguyên.

Code thông thường chưa xử lý ngoại lệ

```
16 public static void main(String[] args) {
17     // TODO code application logic here
18     String st="123a34";
19     int n=Integer.parseInt(st);
20     System.out.print(n);
21 }
```

Với code trên thì quá trình chuyển chuỗi "123a34" sẽ phát sinh ngoại lệ vì ký tự 'a' không là ký số nên quá trình chuyển chuỗi thành số thất bại và xuất hiện Error tại cửa sổ OutPut.

```
run:
Exception in thread "main" java.lang.NumberFormatException: For input string: "123a34"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at ltjava.LTJAVA.main(LTJAVA.java:19)
C:\Users\home\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Để khắc phục ngoại lệ trên ta sử dụng try...catch như sau:

```
16 public static void main(String[] args) {
17     // TODO code application logic here
18     String st="123a34";
19     int n;
20     try{
21         n=Integer.parseInt(st);
22         System.out.print(n);
23     }
24     catch(Exception ex){
25         System.out.print("Không thể chuyển chuỗi thành số!!!!");
26     }
27 }
```

2.10. Một số ví dụ minh họa:

Ví dụ 1: Nhập ký tự từ bàn phím

```
import java.io.*;
```

/ gói này cung cấp thư 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("Ký tự vừa nhập:" + ch);
    }
}
```

Ví dụ 2: Nhập dữ liệu số

```
import java.io.*;
```

```
class inputNum {
    public static void main(String[] args) {
        int n=0;
        try {
            BufferedReader in = new BufferedReader( new InputStreamReader(
                                                                    System.in));

            String s;
            s = in.readLine();
            n = Integer.parseInt(s);
        }
        catch(Exception e){
            System.out.println("Nhập dữ liệu bị lỗi !");
        }
    }
}
```



```
        System.out.println("Bạn vừa nhập số:" + n);
    }
}
```

Ví dụ 3: 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]);
    }
}
```

C:\Oracle\Middleware\Oracle_Home\
This is arrInt[0]: 0
This is arrInt[1]: 1
This is arrInt[2]: 2
This is arrInt[3]: 3
This is arrInt[4]: 4
This is arrInt[5]: 5
This is arrInt[6]: 6
This is arrInt[7]: 7
This is arrInt[8]: 8
This is arrInt[9]: 9
Process exited with exit code 0.

Câu hỏi và bài tập

1. Hãy xác định đâu là tên biến đúng, đâu là tên biến sai?

Area; radius; 5a; readInteger; hoc sinh; sinhVien1; sinhVien2; 1gia tri; if; hoTen; _giaTri; try; a ;
_a; A; _b; _B; \$d

2. Khai báo và gán giá trị cho 2 biến có kiểu số nguyên. In ra màn hình giá trị tổng, hiệu, thương của 2 biến.
3. Khai báo hằng $PI = 3.14$ kiểu số thực, với biến r là bán kính đường tròn – kiểu số thực được gán trong chương trình. Hãy viết chương trình tính diện tích và chu vi hình tròn, in kết quả ra màn hình.
4. Hãy viết chương trình tính tổng các chữ số của một chuỗi số nguyên bất kỳ. Ví dụ: Chuỗi số '8545604' có tổng các chữ số là: $8+5+4+5+6+0+4=32$.
5. Khai báo các biến a, b, c kiểu số nguyên. Gán giá trị cho các biến. Tìm phần nguyên khi chia các số này cho 2, tìm phần dư khi chia các số này cho 3, in kết quả ra màn hình. Tăng giá trị a, b, c mỗi biến lên 1, in giá trị 3 số ra màn hình
6. Giải phương trình $2ax+b = 8c$ với a, b, c là số thực được nhập từ bàn phím. Sau khi tính x , in kết quả ra màn hình, ép kiểu x về số nguyên, in kết quả ép kiểu của x ra màn hình.

7. Cho a, b, c là độ dài 3 cạnh tam giác ($a, b, c > 0$ được nhập từ bàn phím). Tính diện tích s của tam giác, in kết quả ra màn hình. (Gợi ý: Sử dụng công thức Heron với S : diện tích; p : nửa chu vi)
8. Với a, b, c là các số thực được nhập từ bàn phím. Hãy kiểm tra xem đây có phải là độ dài 3 cạnh của 1 tam giác không, nếu có thì đó là của loại tam giác nào?
9. Giải phương trình bậc 2: $ax^2 + bx + c = 0$ với a, b, c là các số thực được nhập từ bàn phím. In kết quả các nghiệm ra màn hình (Nếu $a = 0$ thì ra màn hình đây không phải là phương trình bậc 2 và in nghiệm)
10. Viết chương trình khai báo và nhập biến nguyên a một giá trị bất kỳ. Nếu $a = 1$ thì in ra màn hình là "Chủ nhật", $a = 2$ thì in ra "Thứ Hai", $a = 7$ thì in ra "Thứ Bảy". Nếu a không trong khoảng $[1;7]$ thì báo "Bạn đã sai, chỉ được nhập số nguyên từ 1 tới 7"
11. Khai báo và nhập giá trị cho biến a là 1 trong các số nguyên từ 0 đến 9. Hãy in ra tên các chữ số nhập vào dưới hình thức tiếng Anh. Ví dụ a bằng 1 thì chương trình chạy sẽ in ra "1 đọc là One"; $a = 2$ thì in ra "2 đọc là Two"...
12. Khai báo và nhập giá trị cho 2 biến nguyên "tháng" và "năm". Yêu cầu "tháng" thuộc tập hợp $[1..12]$, năm không được âm. Nếu nhập sai tháng thì báo "Bạn đã nhập sai tháng", nếu nhập sai năm thì báo "Bạn đã nhập sai năm". Khi 1 trong 2 thông tin bị sai thì thoát chương trình, ngược lại:
 - + Nếu năm đó là năm nhuận thì in ra thông báo: Đây là năm nhuận. Không thì báo ra là năm thường.
 - + Dựa vào thông tin năm đó là năm nhuận hay không và giá trị của tháng đó là tháng nào để báo ra tháng đó có bao nhiêu ngày.
13. Hãy khai báo và nhập dữ liệu cho biến n là số nguyên. In ra màn hình bảng cửu chương tương ứng với số vừa nhập.
14. In ra toàn bộ bảng cửu chương từ 1 đến 10 theo 2 dạng:
 - Dạng 1: Các bảng cửu chương in trên các dòng khác nhau
 - Dạng 2: Mỗi bảng cửu chương xếp ở 2 cột
15. Sắp xếp 1 dãy n số nguyên nhập vào từ bàn phím theo thứ tự tăng dần.
16. Nhập các phần tử của 2 ma trận cùng số hàng, số cột vào từ bàn phím. Tính tổng, hiệu, tích 2 ma trận. In kết quả ra màn hình.
17. Nhập vào một chuỗi các số nguyên (mỗi số cách nhau khoảng trắng). Cho biết:
 - a. Chuỗi vừa nhập có bao nhiêu số, đó là những số nào.
 - b. In ra các số là số nguyên tố.
18. Nhập vào một chuỗi các số thực (mỗi số cách nhau dấu chấm phẩy). Cho biết:
 - a. Có bao nhiêu số vừa nhập.
 - b. Hãy làm tròn các số (VD: $2.3 \Rightarrow$ làm tròn là 2; $2.5 \Rightarrow$ làm tròn là 3).
19. Một chuỗi được gọi là số thuận nghịch đọc nếu ta đọc từ trái sang phải hay từ phải sang trái số đó ta vẫn nhận được một số giống nhau.
20. Hãy liệt kê tất cả các số thuận nghịch đọc có 6 chữ số (Ví dụ số: 558855).

- 21.** Viết chương trình liệt kê tất cả các số nguyên tố có 5 chữ số sao cho tổng của các chữ số trong mỗi số nguyên tố đều bằng S cho trước.
- 22.** Viết chương trình nhập vào n số thực dương. Cho biết trong n số đã nhập:
 - Tổng các số dương đã nhập.
 - Số lớn nhất.
 - Các số có phần nguyên nguyên tố.
 - Các số thực có phần thập phân là 0.
- 23.** Viết chương trình nhập vào vào ma trận A có n dòng, m cột, các phần tử là những số nguyên lớn hơn 0 và nhỏ hơn 100 được nhập vào từ bàn phím. Thực hiện các chức năng sau:
 - a. Tìm phần tử lớn nhất của ma trận cùng chỉ số của số đó.
 - b. Tìm và in ra các phần tử là số nguyên tố của ma trận (các phần tử không nguyên tố thì thay bằng số 0).
 - c. Sắp xếp tất cả các cột của ma trận theo thứ tự tăng dần và in kết quả ra màn hình.
- 24.** Viết chương trình nhập vào vào mảng A có n phần tử, các phần tử là những số nguyên lớn hơn 0 và nhỏ hơn 100 được nhập vào từ bàn phím. Thực hiện các chức năng sau:
 - a. Tìm phần tử lớn nhất và lớn thứ 2 trong mảng cùng chỉ số của các số đó.
 - b. Sắp xếp mảng theo thứ tự giảm dần.
 - c. Nhập một số nguyên x và chèn x vào mảng A sao cho vẫn đảm bảo tính sắp xếp giảm dần.
- 25.** Viết chương trình thực hiện chuẩn hoá một xâu ký tự nhập từ bàn phím (loại bỏ các dấu cách thừa, chuyển ký tự đầu mỗi từ thành chữ hoa, các ký tự khác thành chữ thường)
- 26.** Viết chương trình thực hiện nhập một xâu ký tự và tìm từ dài nhất trong xâu đó. Từ đó xuất hiện ở vị trí nào? (Chú ý nếu có nhiều từ có độ dài giống nhau thì chọn từ đầu tiên tìm thấy).

Chương 3: HƯỚNG ĐỐI TƯỢNG TRONG JAVA

3.1. Lớp và đối tượng

Thông qua chuyên đề lập trình hướng đối tượng (OOP) chúng ta đã biết 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. Chương này sẽ giúp bạn đọc tìm hiểu về các kiểu dữ liệu dẫn xuất đó là lớp (class) và giao tiếp (interface), cũng như các vấn đề cơ bản về lập trình hướng đối tượng trong java thông qua việc tạo lập các lớp, các đối tượng và các tính chất của chúng.

3.1.1. Khái niệm lớp

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).

3.1.2. 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.

3.1.3. Tạo đối tượng của lớp

```
ClassName objectName = new ClassName();
```

3.1.4. 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; // so luong so cua xe may: 3, 4 so  
    protected int so;        // sobanhxe là biến tĩnh có giá trị là 2 trong tất cả  
                             // các thể hiện tạo ra từ lớp xemay  
    public static int        sobanhxe = 2;  
}
```

Thuộc tính “nhasx”, “model” có thể được truy cập đến từ tất cả các đối tượng khác.

Thuộc tính “chiphisx” chỉ có thể truy cập được từ các đối tượng có kiểu “xemay”.

Thuộc tính “thoigiansx”, so có thể truy cập được từ các đối tượng có kiểu “xemay” và các đối tượng của các lớp con dẫn xuất từ lớp “xemay”.

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.

3.2. 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.

- **synchronized**: 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 multithreads.

<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 khởi tạo sau khi khai báo.

Với Java thì chỉ hỗ trợ phương thức truyền tham trị không hỗ trợ phương thức truyền tham biến như các ngôn ngữ lập trình khác.

Ví dụ:

```
public class xemay {  
    public String nhax;  
    public String model;  
    private float chiphix;  
    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 * chiphix;  
    }  
}
```

3.2.1. Khởi tạo một đối tượng (Constructor)

Constructor 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 mặc 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;  
}  
}
```

3.2.2. Biến this

Biến **this** là một biến ẩn tồn tại trong tất cả các lớp trong ngôn 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.

Ví dụ:

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

3.3. 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);
}
}
```

3.4. Đặc điểm hướng đối tượng trong 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).

3.4.1. 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.

Ví dụ:

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



```
        var_2 = arr_Object[i];
        var_2. method_1();
    }

}
```

Vòng lặp for trong đoạn chương trình trên:

Với $i = 0$ thì biến `var_2` có kiểu là `B_Object`, và lệnh `var_2. method_1()` sẽ gọi thực hiện phương thức `method_1` của lớp `B_Object`.

Với $i = 1$ thì biến `var_2` có kiểu là `A_Object`, và lệnh `var_2. method_1()` sẽ gọi thực hiện phương thức `method_1` của lớp `A_Object`.

Trong ví dụ trên đối tượng `var_2` có thể nhận kiểu `A_Object` hay `B_Object`. Hay nói cách khác, một biến đối tượng kiểu `A_Object` như `var_2` trong ví dụ trên có thể tham chiếu đến bất kỳ đối tượng nào của bất kỳ lớp con nào của lớp `A_Object` (ví dụ `var_2` có thể tham chiếu đến đối tượng `var_1`, `var_1` là đối tượng của lớp `B_Object` dẫn xuất từ lớp `A_Object`). Ngược lại một biến của lớp con không thể tham chiếu đến bất kỳ đối tượng nào của lớp cha.

3.4.2. Tính kế thừa (inheritance)

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 superclass).

Dùng từ khóa **extends** để chỉ lớp dẫn xuất.

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

3.4.2.1. Khái 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 `tinghiaban()` 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.

3.4.2.2. 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

3.4.2.3. 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() { ... }
}
```

3.4.2.4 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.

3.4.2.5. 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 vì 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.

Ví dụ:

```
class A {
    // Khai báo các thuộc tính public void method_1() { ... }
```

```
protected void finalize() {  
    // Có thể dùng để đóng tất cả các kết nối  
    // vào cơ sở dữ liệu trước khi hủy đối tượng.  
}  
}
```

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.

Ví dụ:

```
package phuongtiengiaothong;  
class xemay { // .... }  
class xega extends xemay { // ... }
```

Khi đó muốn sử dụng lớp *xemay* vào chương trình ta sẽ khai báo như sau:

```
import phuongtiengiaothong. xemay;
```

3.5. Giao tiếp (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 nhax = "Honda VN";  
    static final String dienthoai = "08-8123456";  
    public int gia(String s_model);  
}
```

```
// khai báo 1 lớp có cài đặt interface
public class xemay implements sanpham {
    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
{
    // ...
}
```

3.5.3. Ví dụ minh họa

Ví dụ 1: Minh họa tính đa hình (polymorphism) trong phân cấp kế thừa thông qua việc mô tả và xử lý một số thao tác cơ bản trên các đối tượng hình học.

// Định nghĩa lớp trừu tượng cơ sở tên **Shape** trong

// tập tin Shape.java

```
public abstract class Shape extends Object {
    public double area() { return 0. 0; }
    public double volume(){ return 0. 0; }
    public abstract String getName();
}
```

// Định nghĩa lớp Point trong tập tin Point.java

```
public class Point extends Shape {
    protected int x, y;
    public Point() { setPoint( 0, 0 ); }
    public Point(int xCoordinate, int yCoordinate) { setPoint( xCoordinate, yCoordinate ); }
    public void setPoint( int xCoordinate, int yCoordinate ) {
        x = xCoordinate; y = yCoordinate;
    }
    public int getX() { return x; }
    public int getY() { return y; }
```

```
public String toString() { return "[" + x + ", " + y + "];" ; }  
public String getName() { return "Point"; }  
}
```

Định nghĩa một lớp cha **Shape** là một lớp trừu tượng dẫn xuất từ Object và có 3 phương thức khai báo dùng tiền tố public. Phương thức **getName()** khai báo trừu tượng vì vậy nó phải được hiện thực trong các lớp con. Phương thức **area()** (tính diện tích) và phương thức **volume()** (tính thể tích) được định nghĩa và trả về 0. 0. Những phương thức này sẽ được khai báo chồng trong các lớp con để thực hiện chức năng tính diện tích cũng như thể tích phù hợp với những đối tượng hình học tương ứng (đường tròn, hình trụ, ...)

Lớp **Point**: dẫn xuất từ lớp **Shape**. Một điểm thì có diện tích và thể tích là 0. 0, vì vậy những phương thức **area()** và **volume()** của lớp cha không cần khai báo chồng trong lớp **Point**, chúng được thừa kế như đã định nghĩa trong lớp trừu tượng **Shape**. Những phương thức khác như **setPoint(...)** để gán tọa độ x, y cho một điểm, còn phương thức **getX()**, **getY()** trả về tọa độ x, y của một điểm. Phương thức **getName()** là hiện thực cho phương thức trừu tượng trong lớp cha, nếu như phương thức **getName()** mà không được định nghĩa thì lớp **Point** là một lớp trừu tượng.

// Định nghĩa lớp Circle trong tập tin Circle.java

```
public class Circle extends Point {  
    protected double radius;  
    public Circle() {setRadius( 0 ); }  
    public Circle( double circleRadius, int xCoordinate, int yCoordinate ) {  
        super( xCoordinate, yCoordinate );  
        setRadius( circleRadius );  
    }  
    public void setRadius( double circleRadius ){  
        radius = ( circleRadius >= 0 ?    circleRadius:0 );  
    }  
    public double getRadius() { return radius; }  
    public double area() { return Math. PI * radius * radius; }  
    public String toString() {  
        return "Center = " + super.toString() + "; Radius = " + radius;  
    }  
    public String getName() {return "Circle"; }  
}
```

Lớp **Circle** dẫn xuất từ lớp **Point**, một đường tròn có thể tích là 0. 0, vì vậy phương thức **volume()** của lớp cha không khai báo chồng, nó sẽ thừa kế từ lớp **Point**, mà lớp **Point** thì thừa kế từ lớp **Shape**. Diện tích đường tròn khác với một điểm, vì vậy phương thức tính diện tích **area()** được khai báo chồng. Phương thức **getName()** hiện thực phương thức trừu tượng đã khai báo trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Circle** thì nó sẽ kế thừa từ lớp **Point**. Phương thức **setRadius** dùng để gán một bán kính (radius) mới cho một đối tượng đường tròn, còn phương thức **getRadius** trả về bán kính của một đối tượng đường tròn.

// Định nghĩa lớp hình trụ Cylinder // trong tập tin Cylinder.java.

```
public class Cylinder extends Circle
{
    public Cylinder() { setHeight( 0 ); }
    public Cylinder(double cylinderHeight, double cylinderRadius, int xCoordinate, int
        yCoordinate ) {
        super( cylinderRadius, xCoordinate, yCoordinate );
        setHeight( cylinderHeight );
    }
    public void setHeight( double cylinderHeight ) {
        height = ( cylinderHeight >= 0 ? cylinderHeight : 0 );
    }

    public double getHeight() { return height; }
    public double area() { return 2 * super. area() + 2 * Math. PI * radius * height; }
    public double volume() { return super. area() * height; }
    public String toString() { return super.toString() + "; Height = " + height; }
    public String getName() { return "Cylinder"; }
}
```

Lớp **Cylinder** dẫn xuất từ lớp **Circle**. Một **Cylinder** (hình trụ) có diện tích và thể tích khác với một **Circle** (hình tròn), vì vậy cả hai phương thức **area()** và **volume()** cần phải khai báo chồng. Phương thức **getName()** là hiện thực phương thức trừu tượng trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Cylinder** thì nó sẽ kế thừa từ lớp **Circle**. Phương thức **setHeight** dùng để gán chiều cao mới cho một đối tượng hình trụ, còn phương thức **getHeight** trả về chiều cao của một đối tượng hình trụ.

// Test.java

// Kiểm tra tính kế thừa của Point, Circle, Cylinder với // lớp trừu tượng Shape. // Khai báo thư viện import java. text. DecimalFormat;

```
public class Test {
    public static void main( String args[] ) {
        Point point = new Point( 7, 11 );
        Circle circle = new Circle( 3. 5, 22, 8 );
        Cylinder cylinder = new Cylinder( 10, 3. 3, 10, 10 );
        Shape arrayOfShapes[] = new Shape[ 3 ];
        String output = point.getName() + ": " + point.toString() + "\n" +
            circle.getName() + ": " + circle.toString() + "\n" +
            cylinder.getName() + ": " + cylinder.toString();
        DecimalFormat precision2 = new DecimalFormat( "0. 00" );
        for ( int i = 0; i < arrayOfShapes. length; i++ ) {
            output += "\n\n" + arrayOfShapes[ i ].getName() +
                ": " + arrayOfShapes[ i ].toString() + "\n Area = " +
                precision2. format( arrayOfShapes[ i ]. area() ) +
                "\nVolume = " + precision2. format( arrayOfShapes[ i ]. volume() );
        }
    }
}
```

```
    }  
    System.out.println(output);  
    System. exit( 0 );  
    }  
}
```

Ví dụ 2: Tương tự ví dụ 1 nhưng trong ví dụ 2 chúng ta dùng interface để định nghĩa cho **Shape** thay vì một lớp trừu tượng. Vì vậy tất cả các phương thức trong interface **Shape** phải được hiện thực trong lớp **Point** là lớp cài đặt trực tiếp interface **Shape**.

```
public interface Shape {  
    public abstract double area();  
    public abstract double volume();  
    public abstract String getName();  
}
```

Lớp *Point* cài đặt/hiện thực interface tên *shape*.

```
public class Point extends Object implements Shape {  
    protected int x, y;  
    public Point() { setPoint( 0, 0 ); }  
    public Point(int xCoordinate, int yCoordinate) { setPoint( xCoordinate, yCoordinate ); }  
    public void setPoint( int xCoordinate, int yCoordinate ) {  
        x = xCoordinate; y = yCoordinate;  
    }  
    public int getX() { return x; }  
    public int getY() { return y; }  
    public String toString() { return "[" + x + ", " + y + "]"; }  
    public double area() { return 0. 0; }  
    public double volume() { return 0. 0; }  
    public String getName() { return "Point"; }  
}
```

Lớp *Circle* là lớp con của lớp *Point*, và cài đặt/hiện thực gián tiếp interface tên *shape*.

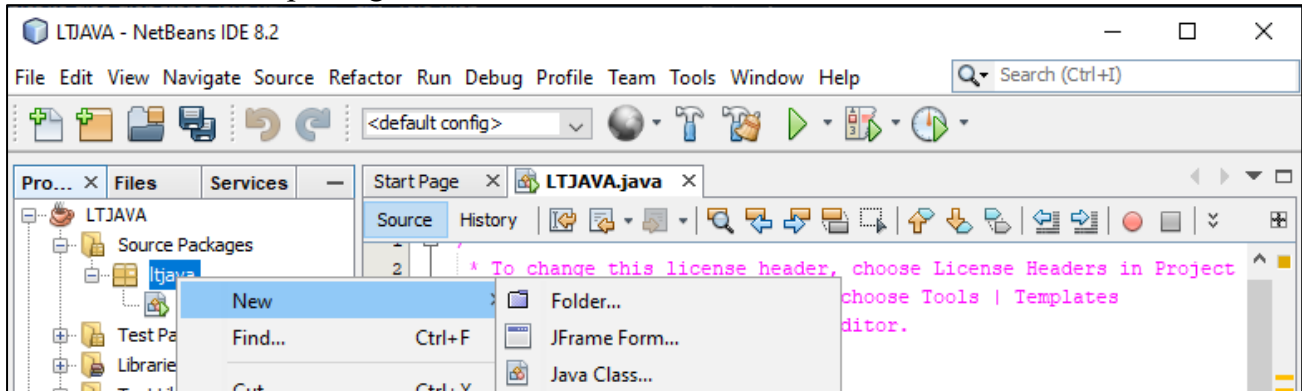
```
public class Circle extends Point {  
    protected double radius;  
    public Circle() { setRadius( 0 ); }  
    public Circle( double circleRadius, int xCoordinate, int yCoordinate ) {  
        super( xCoordinate, yCoordinate );  
        setRadius( circleRadius );  
    }  
    public void setRadius( double circleRadius ){  
        radius = ( circleRadius >= 0 ? circleRadius:0 );  
    }  
    public double getRadius() { return radius; }  
    Circle public double area() { return Math. PI * radius * radius; }  
    public String toString() {
```



```
        return "Center = " + super.toString() + "; Radius = " + radius;
    }
    public String getName() { return "Circle"; }
}
public class Cylinder extends Circle {
    protected double height;
    public Cylinder() { setHeight( 0 ); }
    public Cylinder(double cylinderHeight, double cylinderRadius, int xCoordinate, int
        yCoordinate ) {
        super( cylinderRadius, xCoordinate, yCoordinate );
        setHeight( cylinderHeight );
    }
    public void setHeight( double cylinderHeight ) {
        height = ( cylinderHeight >= 0 ? cylinderHeight :0 );
    }
    public double getHeight() { return height; }
    public double area() { return 2 * super. area() + 2 * Math. PI * radius * height; }
    public double volume() { return super. area() * height; }
    public String toString() { return super.toString() + "; Height = " + height; }
    public String getName() { return "Cylinder"; }
}
public class Test {
    public static void main( String args[] ) {
        Point point = new Point( 7, 11 );
        Circle circle = new Circle( 3. 5, 22, 8 );
        Cylinder cylinder = new Cylinder( 10, 3. 3, 10, 10 );
        Shape arrayOfShapes[] = new Shape[ 3 ];
        Point arrayOfShapes[ 0 ] = point;
        Circle arrayOfShapes[ 1 ] = circle;
        cylinder arrayOfShapes[ 2 ] = cylinder;
        String output = point.getName() + ": " + point.toString() + "\n" +
            circle.getName() + ": " + circle.toString() + "\n" +
            cylinder.getName() + ": " + cylinder.toString();
        DecimalFormat precision2 = new DecimalFormat( "0. 00" );
        for ( int i = 0; i < arrayOfShapes. length; i++ ) {
            output += "\n\n" + arrayOfShapes[ i ].getName() + ": " +
                arrayOfShapes[ i ].toString() + "\n Area = " +
                precision2. format( arrayOfShapes[ i ]. area() ) + "\nVolume = " +
                precision2. format( arrayOfShapes[ i ]. volume() );
        }
        System.out.println(output);
        System. exit( 0 );
    }
}
```

3.6. Tạo class với NetBeans:

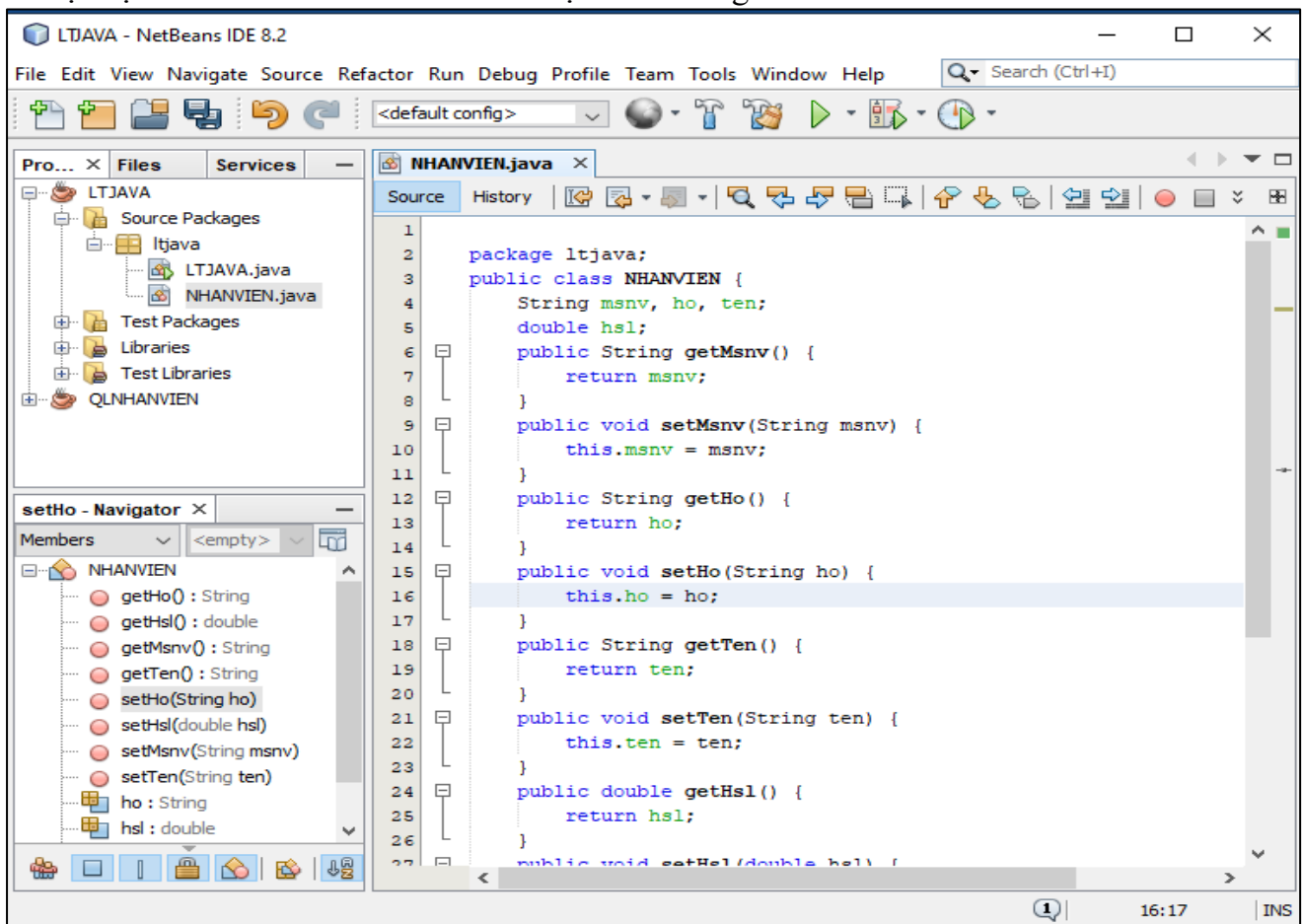
- Xác định Project cần thêm class
- Phải chuột lên package chứa class cần thêm/ New/ Java Class



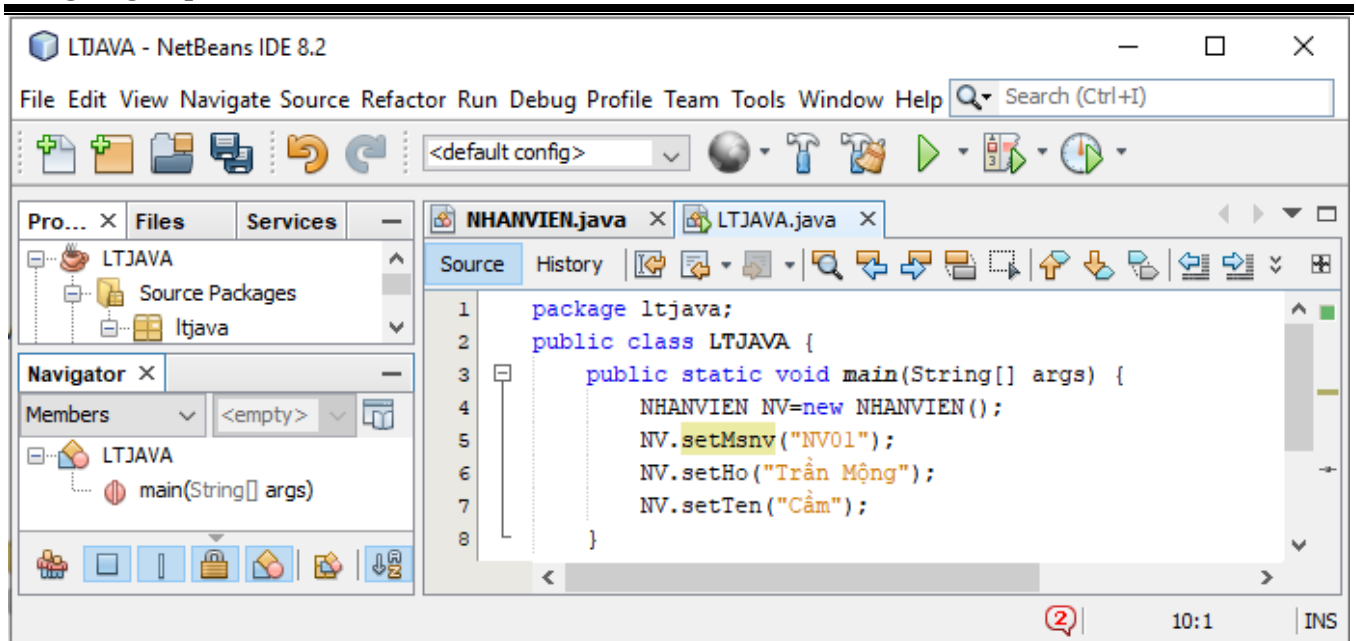
- Đặt tên cho class mới
- Mô tả thuộc tính và xây dựng phương thức cho class

Lưu ý: Chúng ta không thể thực thi class vừa tạo, Muốn minh họa các phương thức trong class thì khai báo đối tượng ở 1 lớp khác hoặc trong hàm main.

Ví dụ: Tạo class NHANVIEN và minh họa class trong hàm main



Hình minh họa class NHANVIEN



Hình ảnh class NHANVIEN được minh họa trong hàm main

CÂU HỎI VÀ BÀI TẬP

1. Xây dựng lớp HCN (hình chữ nhật) có thuộc tính là chiều dài, chiều rộng.

Các phương thức:

- set<thuộc tính>(): gán giá trị cho từng thuộc tính. Ví dụ: phương thức setCD(100) gán giá trị 100 cho thuộc tính chiều dài.
- get<thuộc tính>(): trả về giá trị của các thuộc tính. Ví dụ: phương thức getCD() trả về giá trị hiện tại của thuộc tính chiều dài.
- getCV(): Trả về giá trị chu vi của hình chữ nhật.
- getDT(): Trả về giá trị diện tích của hình chữ nhật.
- Nhap(): để nhập chiều dài, chiều rộng của hình chữ nhật.
- Xuat(): xuất thông tin, chu vi và diện tích hình chữ nhật.

Minh họa việc sử dụng các phương thức của lớp HCN

2. Xây dựng lớp phân số có các thuộc tính: tử, mẫu.

Các phương thức:

- set<thuộc tính>(): gán giá trị cho từng thuộc tính. Ví dụ: phương thức setTu(10) gán giá trị 10 cho thuộc tính tử số.
- get<thuộc tính>(): trả về giá trị của các thuộc tính. Ví dụ: phương thức getTu() trả về giá trị hiện tại của thuộc tính tử số.
- getCong(ps2): Trả về giá trị của phép tính cộng của phân số hiện tại với phân số ps2.
- getTru(ps2): Trả về giá trị của phép tính trừ của phân số hiện tại với phân số ps2.
- getNhan(ps2): Trả về giá trị của phép tính nhân của phân số hiện tại với phân số ps2.
- getChia(ps2): Trả về giá trị của phép tính chia của phân số hiện tại với phân số ps2.
- Nhap(): để nhập tử số và mẫu số.
- Xuat(): xuất thông tin của phân số.

Minh họa cách sử dụng lớp phân số

- Khai báo 2 phân số
 - Nhập và xuất các thông tin của 2 phân số
 - In ra kết quả các phép tính của 2 phân số
 - Xây dựng phương thức Cong để thực hiện cộng 2 phân số.
 - In ra phân số có giá trị lớn hơn trong 2 phân số.
3. Xây dựng lớp **HocSinh** gồm 3 thuộc tính: Mã số học sinh, họ tên, lớp, điểm trung bình. Các phương thức bao gồm:
- set<thuộc tính>(): gán giá trị cho từng thuộc tính. Ví dụ: phương thức setMaHS("HS01") gán giá trị "HS01" cho thuộc tính mã số học sinh.
 - get<thuộc tính>(): trả về giá trị của các thuộc tính. Ví dụ: phương thức getMaHS() trả về giá trị hiện tại của thuộc tính mã số học sinh.
 - Nhap(): Cho phép nhập các thông tin của học sinh từ bàn phím.
 - Xuat(): Xuất các thông tin của học sinh ra màn hình.
 - XepLoai(): Dựa vào điểm trung bình của học sinh mà xác định xếp loại.

Minh họa cách sử dụng lớp HocSinh:

- Tạo 2 đối tượng học sinh hs1 và hs2. Sử dụng phương thức Nhap() để nhập giá trị các thuộc tính cho 2 đối tượng trên. So sánh điểm trung bình giữa 2 đối tượng tìm ra người có điểm cao hơn in ra màn hình.
 - Khai báo danh sách có n (n>0 nhập từ bàn phím) học sinh và thực yêu cầu trên danh sách:
 - Nhập danh sách n học sinh.
 - Xuất danh sách n học sinh.
 - Xuất danh sách của học sinh thuộc lớp "8A-6"
 - Xuất danh sách học sinh theo thứ tự tăng dần theo tên.
 - Xuất danh sách học sinh theo từng lớp.
 - Sắp xếp danh sách giảm dần theo điểm trung bình
 - Xếp hạng cho học sinh theo các lớp.
 - Xuất thông tin của 3 học sinh giỏi nhất ở từng lớp.
 - Xếp hạng cho các học sinh ở từng lớp.
 - Thống kê số lượng học sinh của từng lớp theo từng xếp loại.
 - Sao chép danh sách học sinh ở các lớp khác nhau vào các mảng khác nhau.
4. Xây dựng lớp THISINH (thí sinh) gồm: Số báo danh, họ, tên, năm sinh, điểm toán, lý, hóa, phòng thi.
- Các phương thức của lớp:
- set<thuộc tính>(): gán giá trị cho từng thuộc tính. Ví dụ: phương thức setMaTS("TS01") gán giá trị "TS01" cho thuộc tính số báo danh.
 - get<thuộc tính>(): trả về giá trị của các thuộc tính. Ví dụ: phương thức getMaTS() trả về giá trị hiện tại của thuộc tính mã số học sinh.
 - DTB(): trả về điểm trung bình của thí sinh

- KetQua(): Nếu điểm trung bình từ 5 và không môn nào bị 0 thì kết quả là “Đạt”, ngược lại kết quả “Không đạt”.
- Nhap(): Cho phép nhập các thông tin của thí sinh từ bàn phím.
- Xuat(): Hiển thị thông tin, điểm trung bình và kết quả.

Minh họa cách sử dụng lớp THISINH:

- Tạo danh sách có n thí sinh. $n > 0$ được nhập từ bàn phím.
 - Nhập thông tin cho n thí sinh.
 - Xuất danh sách các thí sinh theo từng phòng thi.
 - Thống kê số lượng thí sinh đạt, không đạt ở mỗi phòng.
 - Thí sinh có tổng điểm lớn nhất
5. Xây dựng lớp hàng hóa gồm: tên hàng, mã hàng, giá bán, số lượng, giờ mua. Nếu mua vào khoảng 8h -> 17h hàng ngày, được giảm giá 5%. Tạo phương thức để nhập, xuất thông tin và phương thức để tính tiền.

Khai báo một danh sách n hàng hóa ($n > 0$, nhập từ bàn phím). Nhập dữ liệu cho danh sách và cho biết tổng tiền.

6. Xây dựng lớp sinh viên với các thuộc tính: hoTen, namSinh, lop, diemTb.

Tạo 1 constructor với 3 tham số truyền vào; 1 constructor không có tham số truyền vào.

Xây dựng phương thức nhập, xuất thông tin cho lớp.

Viết hàm main minh họa.

7. Xây dựng lớp Hình có 1 thuộc tính là cạnh, 1 constructor không đối số, 1 phương thức nhập và xuất. Xây dựng phương thức tính diện tích theo yêu cầu sau:

- Có 1 tham số truyền vào, nó sẽ tự hiểu là cần tính diện tích hình vuông
- Có 2 tham số truyền vào, nó tự hiểu là tính diện tích hình chữ nhật
- Có 3 tham số truyền vào, nó tự hiểu là tính diện tích hình tam giác.

Viết hàm main cho phép lựa chọn từ 1-3 tương ứng tính diện tích hình vuông, chữ nhật, tam giác; đồng thời yêu cầu nhập vào độ dài cạnh để tính diện tích. Sau đó in kết quả ra màn hình.

8. Xây dựng lớp hàng hóa gồm các thuộc tính: tên hàng, giá, số lượng, % giảm giá. Xây dựng constructor không tham số, phương thức nhập và xuất thông tin.

Tạo phương thức tính tiền tổng hóa đơn hàng hóa.

Tạo phương thức nạp chồng phương thức trên, có thêm tham số truyền vào nữa là % giảm giá. Tính tiền dựa vào các thông số này.

Viết CTC Tạo danh sách n hàng hóa, nhập thông tin, tính tiền và in tất cả thông tin ra màn hình (bao gồm tổng số tiền)

9. Tạo package nhân sự, trong có 2 class là sinh viên và giảng viên

Class sinh viên gồm các thuộc tính: họ tên, lớp học, điểm toán, lý, hóa. Các phương thức nhập, xuất thông tin, tính điểm trung bình.

Class giảng viên gồm các thuộc tính: họ tên, năm sinh, chuyên môn, lớp dạy. Các phương thức nhập, xuất thông tin.

Tạo 2 danh sách n sinh viên và m giảng viên ($n, m > 0$, nhập từ bàn phím). Nhập thông tin cho 2 danh sách và thực hiện yêu cầu sau:

- Nhập tên sinh viên. Cho biết những giáo viên nào có dạy sinh viên này.
- Nhập tên giáo viên và lớp dạy. in ra màn hình danh sách sinh viên của lớp mà giáo viên này dạy.
- Cho biết sinh viên có điểm trung bình cao nhất và những giáo viên nào có dạy sinh viên này.

10. Dùng tính kế thừa khai báo 3 class:

Class "Sinh Viên" gồm các thuộc tính: Họ tên, năm sinh, mã thẻ, tiền học phí còn nợ. Class "Giảng Viên" gồm các thuộc tính: Họ tên, năm sinh, mã thẻ, tiền lương hàng tháng.

Class "Giám Đốc" gồm các thuộc tính: Họ tên, năm sinh, mã thẻ, tiền tiêu hàng tháng.

Xây dựng các phương thức getter và setter để lấy và gán giá trị cho các thuộc tính.

Xây dựng hàm main thực hiện các công việc sau:

- a. Tạo 1 đối tượng Sinh viên, nhập giá trị vào từ bàn phím, in thông tin ra màn hình.
 - b. Khai báo một danh sách chứa n Giảng viên ($n > 0$, nhập từ bàn phím). Nhập dữ liệu cho danh sách và in ra danh sách đã được sắp xếp tăng dần theo tên.
 - c. Khai báo một danh sách chứa n Giám đốc ($n > 0$, nhập từ bàn phím) và nhập dữ liệu cho danh sách này. Cho biết những giám đốc nào có tiền tiêu hàng tháng cao nhất trong danh sách vừa nhập.
- 11. Một đơn vị sản xuất gồm có các cán bộ là công nhân, kỹ sư, nhân viên.**
Mỗi cán bộ cần quản lý các thuộc tính: Họ tên, năm sinh, giới tính, địa chỉ.
- Các công nhân cần quản lý: Bậc (công nhân bậc 3/7, bậc 4/7 ...).
 - Các kỹ sư cần quản lý: Ngành đào tạo.
 - Các nhân viên phục vụ cần quản lý thông tin: công việc.
- a. Xây dựng các lớp NhanVien, CongNhan, KySu kế thừa từ lớp CanBo.
 - b. Xây dựng các hàm để truy nhập, hiển thị thông tin và kiểm tra về các thuộc tính của các lớp.

Xây dựng lớp QLCB cài đặt các phương thức thực hiện các chức năng sau:

- Nhập thông tin mới cho cán bộ
- Tìm kiếm theo họ tên
- Hiển thị thông tin về danh sách các cán bộ

12. Các thí sinh dự thi đại học bao gồm các thí sinh thi khối A, thí sinh thi khối B, thí sinh thi khối C

- Các thí sinh cần quản lý các thuộc tính: Số báo danh, họ tên, địa chỉ, ưu tiên.

- Thí sinh thi khối A thi các môn: Toán, lý, hoá
 - Thí sinh thi khối B thi các môn: Toán, Hoá, Sinh
 - Thí sinh thi khối C thi các môn: văn, Sử, Địa
- a. Xây dựng các lớp để quản lý các thí sinh sao cho sử dụng lại được nhiều nhất.
 - b. Xây dựng lớp TuyenSinh cài đặt các phương thức thực hiện các nhiệm vụ sau:
 - Nhập thông tin về các thí sinh dự thi
 - Hiển thị thông tin về một thí sinh
 - Tìm kiếm theo số báo danh
- 13.** Để quản lý hồ sơ học sinh của trường THPT, người ta cần quản lý những thông tin như sau:
- Các thông tin về: lớp, khoá học, kỳ học, và các thông tin cá nhân của mỗi học sinh.
 - Với mỗi học sinh, các thông tin cá nhân cần quản lý gồm có: Họ và tên, tuổi, năm sinh, quê quán.
- a. Hãy xây dựng lớp Nguoi để quản lý các thông tin cá nhân của mỗi học sinh.
 - b. Xây dựng lớp HSHocSinh (hồ sơ học sinh) để lý các thông tin về mỗi học sinh.
 - c. Xây dựng các phương thức: nhập, hiển thị các thông tin về mỗi cá nhân.
 - d. Cài đặt chương trình thực hiện các công việc sau:
 - Nhập vào một danh sách gồm n học sinh (n- nhập từ bàn phím)
 - Hiển thị ra màn hình tất cả những học sinh sinh năm 1985.
 - Cho biết có bao nhiêu học sinh sinh năm 1985 và có quê ở Thái Nguyên.
- 14.** Xây dựng một interface có tên là HCNInterface chứa phương thức sau:
- dientichHCN(); getChieuDai() và getChieuRong(); setDaiRong(cd, cr) Sử dụng HCNInterface trên để xây dựng lớp Hinhchunhat chứa hai thuộc tính là: chieudai, chieurong và triển khai các phương thức trong HCNInterface trên?
- Xây dựng lớp HCNTTest thừa kế lớp Hinhchunhat chứa phương thức main thực hiện các công việc sau:
- a. Khai báo một mảng chứa n hình chữ nhật ($n > 0$, nhập từ bàn phím). Sau đó nhập chiều dài và chiều rộng cho n hình chữ nhật đó.
 - b. In ra màn hình thông tin: chiều dài, chiều rộng và diện tích của n hình chữ nhật.
 - c. In ra màn hình thông tin về hình chữ nhật có diện tích lớn nhất.

Chương 4: GIAO DIỆN NGƯỜI DÙNG

4.1. Tổng quan

Chương này cung cấp cho sinh viên những kiến thức cơ bản để xây dựng giao diện (Graphic User Interface - GUI) của chương trình ứng dụng bằng ngôn ngữ java:

- Những nguyên tắc thiết kế giao diện.
- Những thư viện, gói xây dựng giao diện: gồm những lớp (class), những giao tiếp (interface) quản lý sự kiện và những thành phần (components) xây dựng nên giao diện người dùng.
- Bộ quản lý trình bày (layout managers)
- Xử lý sự kiện

Trong khuôn khổ giáo trình lập trình java căn bản này chúng tôi trình bày việc thiết kế GUI dùng thư viện awt (abstract windows toolkit). Việc thiết kế GUI sẽ trực quan, uyển chuyển hơn khi chúng ta sử dụng thư viện JFC (Java Foundation Class) sẽ giới được giới thiệu trong chuyên đề java nâng cao.

4.2. Thiết kế giao diện người dùng

Thư viện giao diện phổ biến được dùng là awt và Swing 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 giao diện. Nhưng khác với các hàm API, thư viện giao diện không phụ thuộc hệ điều hành. Thư viện giao diện 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)



4.3. Các khái niệm cơ bản

4.3.1. 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

4.3.2.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 sổ window hẵn hoi ở mức trên cùng bao gồm một tiêu đề 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**

- **ScrollPanels:** 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**

4.3.3. 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 này 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.

4.4. Thiết kế GUI cho chương trình

4.4.1. 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[]) {
        Frame fr = new Frame("My First Window");
        fr.setBounds(0, 0, 640, 480);
        fr.setVisible(true);
    }
}
```

4.4.2. Tạo hệ thống thực đơn

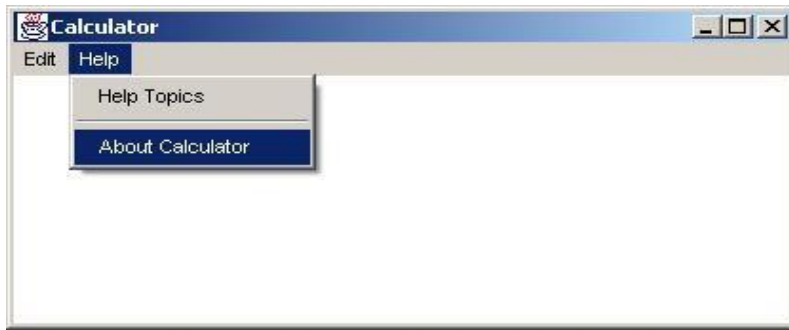
Đối với thư viện awt, để xây dựng hệ thống thực đơn cho chương trình ứng dụng chúng ta có thể dùng các lớp MenuBar, Menu, MenuItem, MenuShortcut.

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() {
        final Frame fr = new Frame();
        fr. setLayout(new BorderLayout());
        MenuBar menu = new MenuBar();
        Menu menuFile = new Menu("File");
        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. setVisible(true);
        fr.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent e) {System. exit(0); }});
    }
}
```

Kết quả thực thi chương trình:



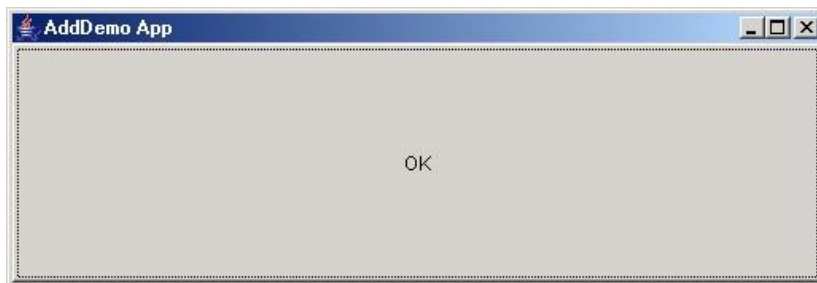
4.4.3. Gắn Component vào khung chứa

Để gắn một thành phần, một đối tượng component vào một cửa sổ (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[]) {
        Frame fr = new Frame("AddDemo App");
        Button buttOk = new Button("OK");
        fr.add(buttOk);
        fr.setSize(100, 100);
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.4. 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

4.4.4.1 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 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));
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.4.2. 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

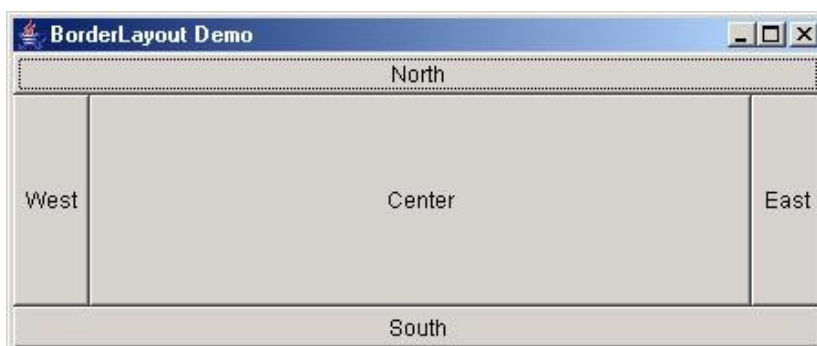
đổi kích thước của những components chứa trong nó sao cho vừa với 5 vùng ĐÔNG, TÂY, NAM, BẮC, TRUNG TÂM.

- 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 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:



4.4.4.3. 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);  
    }  
}
```

4.4.4.4. 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.
 - o **gridx, gridy**: vị trí ô của khung lưới vô hình mà ta sẽ đưa đối tượng con vào
 - o **gridwidth, gridheight**: kích thước hay vùng trình bày cho đối tượng con.
 - o **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).
 - o **weightx, weighty**: chỉ định khoảng cách lớn ra tương đối của các đối tượng con với nhau
- ví dụ: Thiết kế giao diện như hình bên dưới:



4.4.4.5. Null Layout

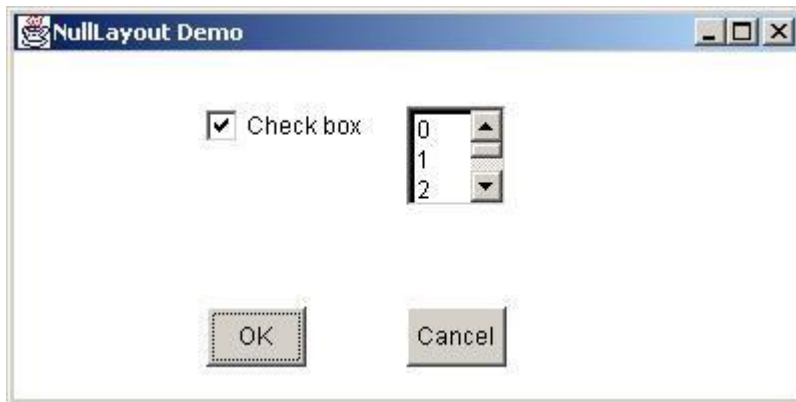
Một khung chứa được trình bày theo kiểu Null Layout có nghĩa là người lập trình phải tự làm tất cả từ việc qui định kích thước của khung chứa, cũng như kích thước và vị trí của từng đối tượng component trong khung chứa.

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:

- o `Public void setLocation(Point p)`
- o `Public void setSize(Dimension p)`
- o `Public void setBounds(Rectangle r)`

Thiết kế giao diện như hình bên dưới.



4.4.5. 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 một 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.

4.4.5.1. Khung chứa Frame

[java.lang. Object](http://java.lang.Object) +-- [java. awt.component](http://java.awt.component)

+--[java.awt.container](#)

+--[java.awt.Window](#)

+--**java.awt.Frame**

Khung chứa Frame là một cửa sổ window hẵn hoi ở mức trên cùng bao gồm một tiêu đề 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.

4.4.5.2. Khung 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.

Ví dụ:

```
import java.awt.*;

public class PanelDemo extends Frame {
    private Button next, prev, first;
    private List li;
    public PanelDemo(String sTitle) {
        super(sTitle);
        next = new Button("Next >>");
        prev = new Button("<< Prev");
        first = new Button("First");
        Panel southPanel = new Panel();
        southPanel.add(next);
        southPanel.add(prev);
        southPanel.add(first);
        this.add(southPanel, BorderLayout.SOUTH);
        Panel northPanel = new Panel();
        northPanel.add(new Label("Make a Selection"));
        this.add(northPanel, BorderLayout.NORTH);
        li = new List();
        for(int i=0; i<10; i++) { li.add("Selection" + i); }
        this.add(li, BorderLayout.CENTER);
    }
    public static void main(String arg[]) {
        Container f = new PanelDemo("Panel Demo");
        f.setSize(300, 200);
    }
}
```

```
        f.setVisible(true);  
    }  
}
```

4.4.5.3. 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 sổ khác của ứng dụng khi dialog dạng này còn hiển thị.

Non-Modal Dialog: vẫn có thể đến các cửa sổ khác của ứng 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:

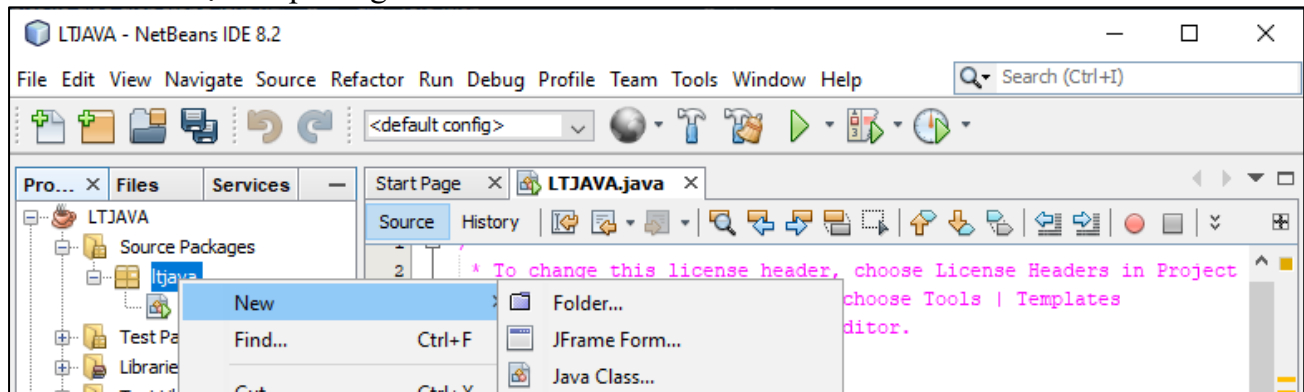
public Dialog (Frame parentWindow, boolean isModal) *public Dialog (Frame parentWindow, String title, boolean isModal)* *parentWindow:* cửa sổ cha *title:* tiêu đề của Dialog *isModal:* true -> là Dialog dạng modal *isModal:* false -> là Dialog không phải dạng modal (hay non-modal)

4.4.5.4. Lập trình giao diện với NetBeans

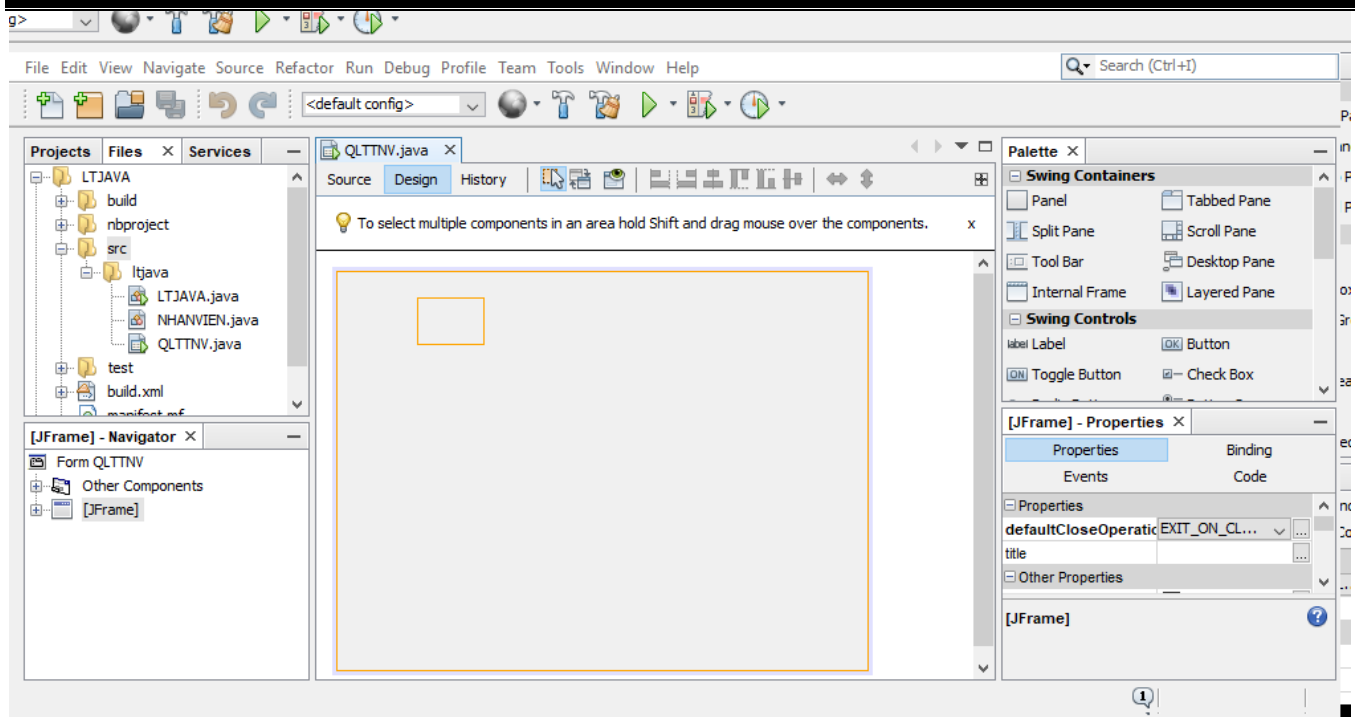
Khai thác NetBeans để thiết kế giao diện người dùng thì công việc thực hiện khá đơn giản thông qua các thao tác kéo thả bằng chuột.

Cách thêm 1 JFrame vào Project

- Xác định Project cần thêm JFrame Form
- Phải chuột lên package chứa class cần thêm/ New/ JFrame Form

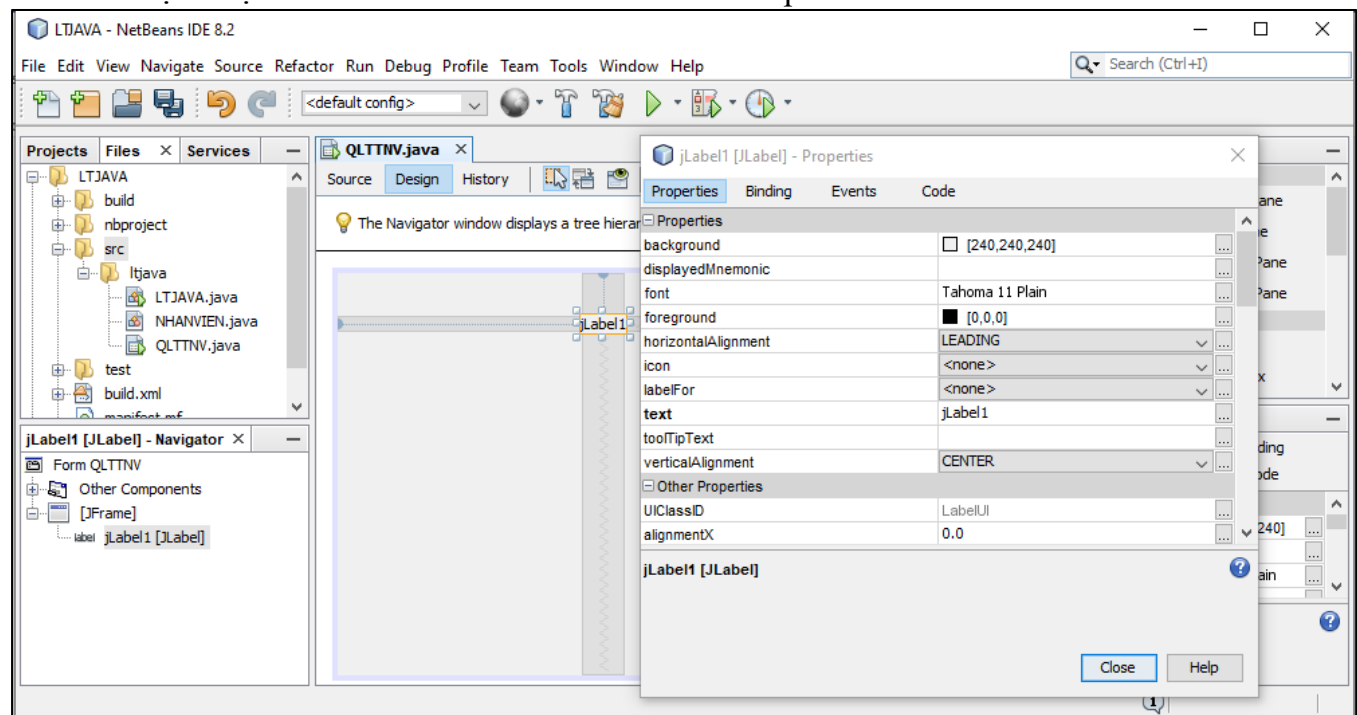


- Đặt tên cho JFrame Form mới
- Sau khi tạo JFrame Form thành công ta được giao diện như sau:



Giao diện mới tạo JFrame Form

- Công việc tiếp theo là chọn các controls phù hợp từ Swing Controls để đặt lên JFrame Form.
- Cài đặt thuộc tính cho các controls trên cửa sổ Properties



Hình ảnh các thuộc tính của JLabel

Khi JFrame Form vừa được tạo thì được giao diện gồm 3 table như hình trên: Source, Design và History.

Source: Xây dựng code sự kiện cho các controls.

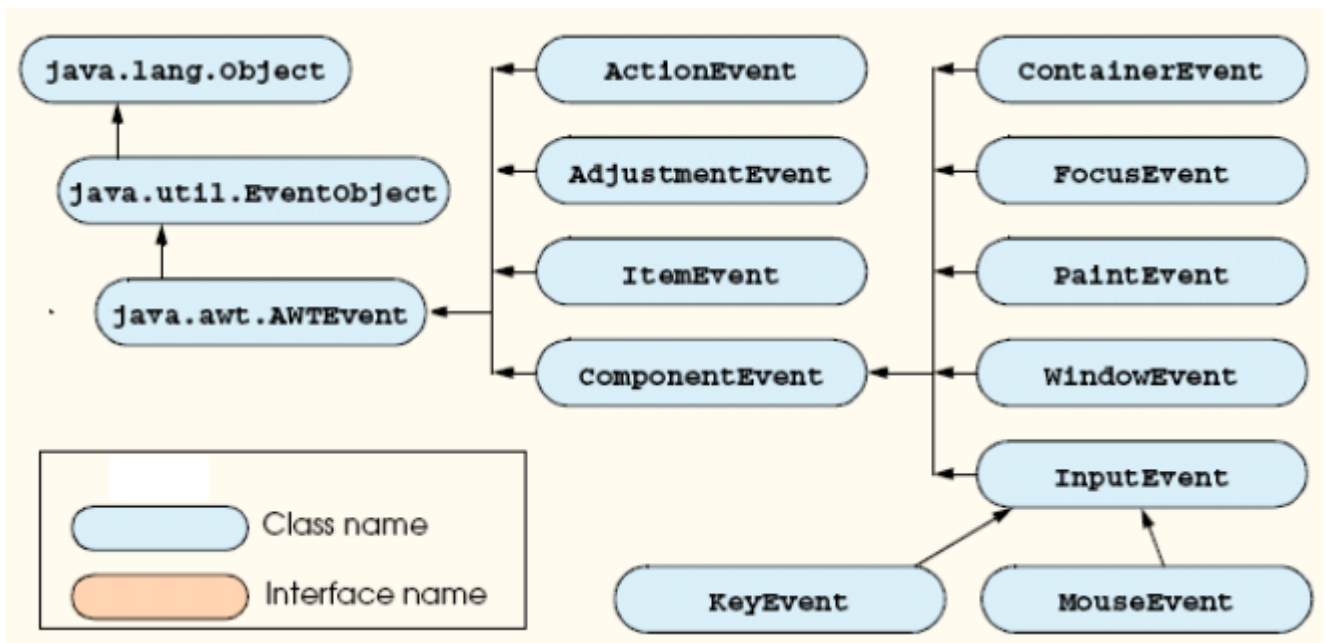
Design: Kéo thả/ bố trí lại các controls lên JFrame Form

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 gửi 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 Java



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 gửi 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 interfaces 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 nhấn và 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).

Ví dụ: Chương trình tên MouseTracker bên dưới minh họa việc dùng những phương thức của các interfaces MouseListener và MouseMotionListener để “bắt” và xử lý các sự kiện chuột tương ứng.

```
import java. awt.*;
import java. awt. event.*;
public class MouseTracker extends Frame implements MouseListener, MouseMotionListener {
    private Label statusBar;
    public MouseTracker() {
        super( "Demonstrating Mouse Events" );
        statusBar = new Label();
        this.add( statusBar, BorderLayout. SOUTH );
        addMouseListener( this );
        addMouseMotionListener( this );
        setSize( 275, 100 );
        setVisible( true );
    }
    public void mouseClicked( MouseEvent event ) {
        statusBar. setText( "Clicked at [" + event.getX() + ", " + event.getY() + "]" );
    }

    public void mousePressed( MouseEvent event ) {
        statusBar. setText( "Pressed at [" + event.getX() + ", " + event.getY() + "]" );
    }
}
```

```
public void mouseReleased( MouseEvent event ) {
    statusBar.setText( "Released at [" + event.getX() + ", " + event.getY() + "]" );
}

public void mouseEntered( MouseEvent event ) {
    statusBar.setText( "Mouse in window" );
}

public void mouseExited( MouseEvent event ){
    statusBar.setText("Mouse outside window" );
}

public void mouseDragged( MouseEvent event ) {
    statusBar.setText( "Dragged at [" + event.getX() + ", " + event.getY() + "]" );
}

public void mouseMoved( MouseEvent event ) {
    statusBar.setText( "Moved at [" + event.getX() + ", " + event.getY() + "]" );
}

public static void main(String args[]){MouseTracker application = new MouseTracker();}
}
```

Kết quả thực thi chương trình:

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*.

Ví dụ: minh họa việc xử lý sự kiện chuột thông qua các phương thức của interface *KeyListener*. Lớp *KeyDemo* bên dưới hiện thực interface *KeyListener*, vì vậy tất cả 3 phương thức trong *KeyListener* phải được cài đặt trong chương trình.

```
import java. awt. *;
```

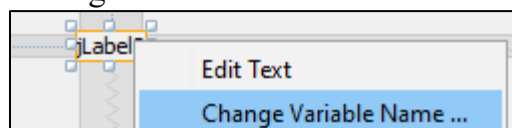
```
import java. awt. event. *;
```

```
public class KeyDemo extends Frame implements KeyListener {
    private String line1 = "", line2 = "", line3 = "";
    private TextArea textArea;
    public KeyDemo() {
        super( "Demonstrating Keystroke Events" );
        textArea = new TextArea( 10, 15 );
        textArea. setText( "Press any key on the keyboard. . ." );
        textArea. setEnabled( false );
        this.add( textArea );
        addKeyListener( this );
        setSize( 350, 100 );
        setVisible( true );
    }
    public void keyPressed( KeyEvent event ) {
        line1 = "Key pressed: " + event.getKeyText( event.getKeyCode() );
        setLines2and3( event );
    }
    public void keyReleased( KeyEvent event ) {
        line1 = "Key released: " + event.getKeyText( event.getKeyCode() );
        setLines2and3( event );
    }
    public void keyTyped( KeyEvent event ) {
        line1 = "Key typed: " + event.getKeyChar();
        setLines2and3( event );
    }
    private void setLines2and3( KeyEvent event ) {
        line2 = "This key is " + ( event. isActionKey() ? "" : "not " ) + "an action key";
        String temp = event.getKeyModifiersText( event.getModifiers() );
        line3 = "Modifier keys pressed: " + ( temp. equals( "" ) ? "none" : temp );
        textArea. setText( line1 + "\n" + line2 + "\n" + line3 + "\n" );
    }
    public static void main( String args[] ) { KeyDemo application = new KeyDemo(); }
}
```

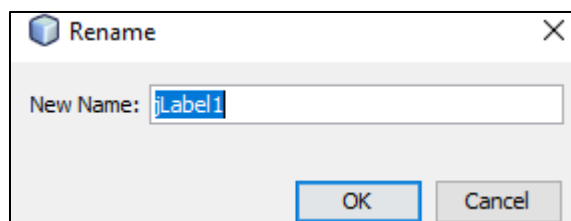

4.5.4. Tạo sự kiện cho các controls trên NetBeans

Trước khi xây dựng code cho các controls trên JFrame Form thì các control cần được đặt tên hợp lý theo cách sau:

- Phải chuột lên control/ Change Variable Name

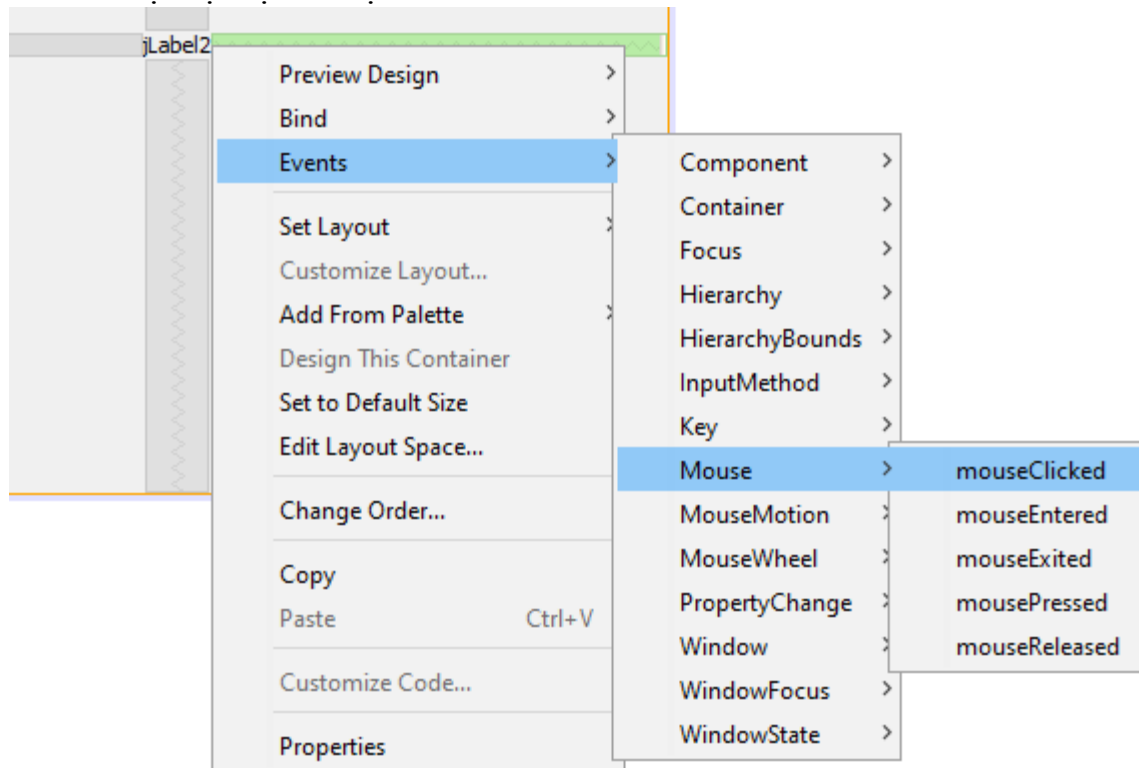


- Đặt tên cho control



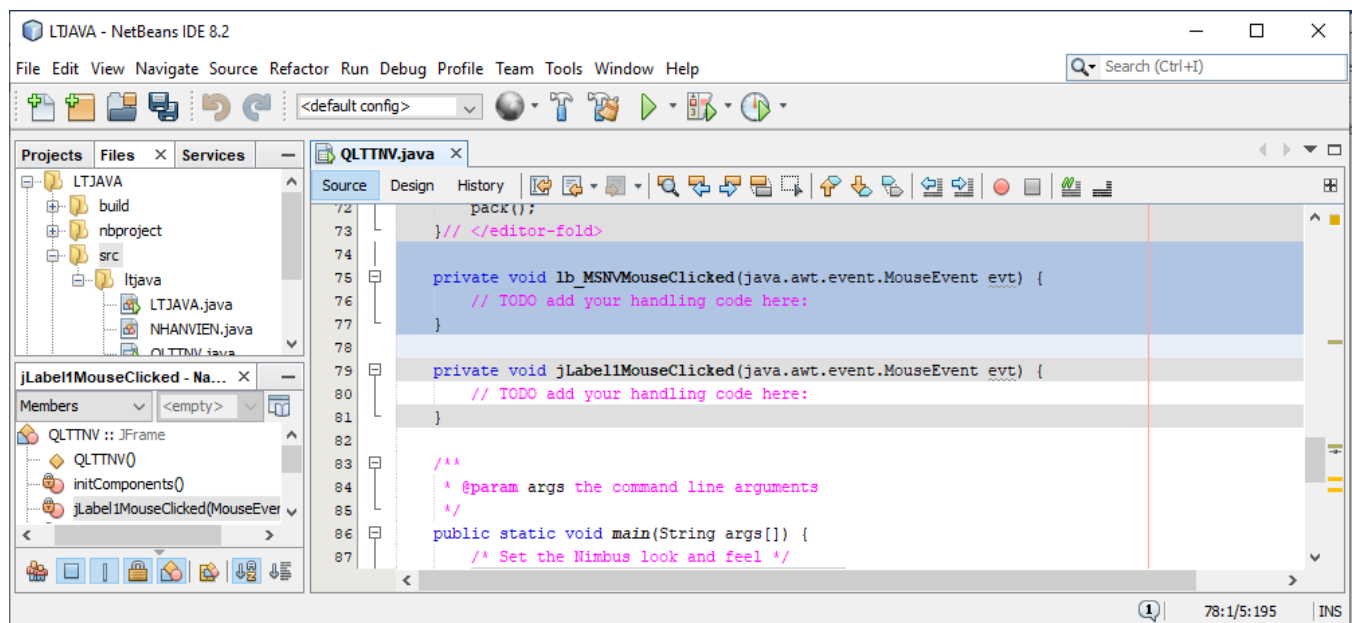
Cách tạo sự kiện cho controls trên NetBeans

- Phải chuột lên controls cần tạo sự kiện/ Event
- Chọn sự kiện cài đặt



Giao diện chọn sự kiện mouseClicked cho control JLabel

- Giao diện của tab Source sẽ xuất hiện để người lập trình xây dựng code sự kiện cho control



Giao diện sự kiện MouseClick của JLabel có name là lb_MSNNV

CÂU HỎI VÀ BÀI TẬP

1. Thiết kế form có 1 label như sau như sau



Khi người dùng click vào label có nội dung “HELLO EVERY BODY!!!” thì nội dung đổi thành “My name’s <tên SV>”.

2. Thiết kế giao diện như sau



Thực hiện tính chu vi, diện tích hình chữ nhật khi người dùng click vào button tương ứng.

3. Viết chương trình xây dựng giao diện thực hiện chức năng: nhập; cộng; trừ; nhân chia 2 phân số.
4. Viết chương trình xây dựng giao diện “máy tính cá nhân bao gồm: cộng, trừ, nhân, chia”.
5. Viết chương trình xây dựng giao diện giải phương trình bậc một.
6. Viết chương trình xây dựng giao diện giải phương trình bậc hai
7. Cải tiến lại tập 2 với giao diện cho phép lựa chọn giải phương trình bậc một, bậc hai.
8. Viết chương trình xây dựng giao diện “Lựa chọn tính chu vi, diện tích hình tròn và hình chữ nhật”.
9. Viết chương trình xây dựng giao diện “chương trình nhập, xuất, sắp xếp tăng dần trong mảng 1 chiều”.
10. Viết chương trình xây dựng giao diện “Nhập 2 số, Tìm ước chung lớn nhất, bội chung nhỏ nhất của 2 số”.
11. Thiết kế giao diện máy tính như Calculator của Windows để thực hiện các phép toán cơ bản.
12. Thiết kế giao diện nhập mảng như sau:

13. Viết chương trình xây dựng giao diện như sau:

14. Xây dựng ứng dụng quản lý thông tin nhân viên theo mẫu:

- a) Khi người dùng chọn nút “Đăng ký” tất cả thông tin đã nhập và chọn sẽ được hiển thị ra màn hình. Nếu người dùng chưa nhập hoặc chưa chọn, ứng dụng sẽ hiển thị thông báo yêu cầu người dùng hoàn nhập và chọn thông tin nào còn thiếu.
- b) Khi người dùng chọn nút “Bỏ qua”, thiết lập nội dung hiển thị của các thành phần trên màn hình giống như lúc ban đầu.

15. Xây dựng class SinhVien để mô tả sinh viên gồm có:

Thuộc tính: Mã số sinh viên, họ tên, ngày sinh, điểm trung bình, giới tính, tên lớp.

Phương thức:

- Các phương thức set<thuộc tính>: Gán giá trị cho thuộc tính. Ví dụ phương thức setMaSV("1CTT20A1"): gán giá trị "1CTT20A1" vào cho thuộc tính mã số sinh viên. Tương tự xây dựng các phương thức set<thuộc tính> khác để gán giá trị cho các thuộc tính khác.
- Các phương thức get<<Thuộc tính>>: nhận về giá trị của các thuộc tính. Ví dụ getHoTen(): trả về giá trị của thuộc tính họ tên.
- getXepLoai(): trả về loại tương ứng với điểm trung bình của sinh viên.
- getTuoi(): trả về tuổi của sinh viên dựa vào ngày sinh.

16. Thiết kế giao diện như sau:

The screenshot shows a Java Swing window titled "DANH SÁCH SINH VIÊN". It features several input fields: "Mã số" (empty), "Họ tên" (empty), "Ngày sinh" (empty), "Điểm TB" (containing the value "5"), and "Lớp" (a dropdown menu currently showing "CNTT 2019"). There is also a checkbox labeled "Nam" which is checked. Below these fields are four buttons: "XÁC NHẬN", "ADD MẢNG", "NHẬP LẠI", and "THOÁT". At the bottom left of the window, the text "Thông tin sinh viên" is displayed.

Yêu cầu:

Text field Mã số sinh viên: Nhập đủ 8 ký tự.

Text field Họ tên: Nhập ít nhất 2 từ.

Check box: được chọn thì hiển thị Nam ngược lại hiển thị nữ

Text field Ngày sinh: Nhập tuổi không dưới 18.

Text field Điểm trung bình: Nhập từ 0 đến 10

Combo Lớp: chọn từ danh sách

Button XÁC NHẬN: Khi click vào thì in thông tin của sinh viên ở cuối form.

Button ADD MẢNG: Thêm thông tin hiện tại vào mảng chứa danh sách sinh viên.

Button NHẬP LẠI: Xóa các thông tin hiện tại trên form.

Button ĐÓNG: Đóng form hiện tại.

17. Thiết kế giao diện để thực hiện các phép toán : '+', '-', '*', ':', '%', '^'.

a) Thiết kế giao diện như hình.

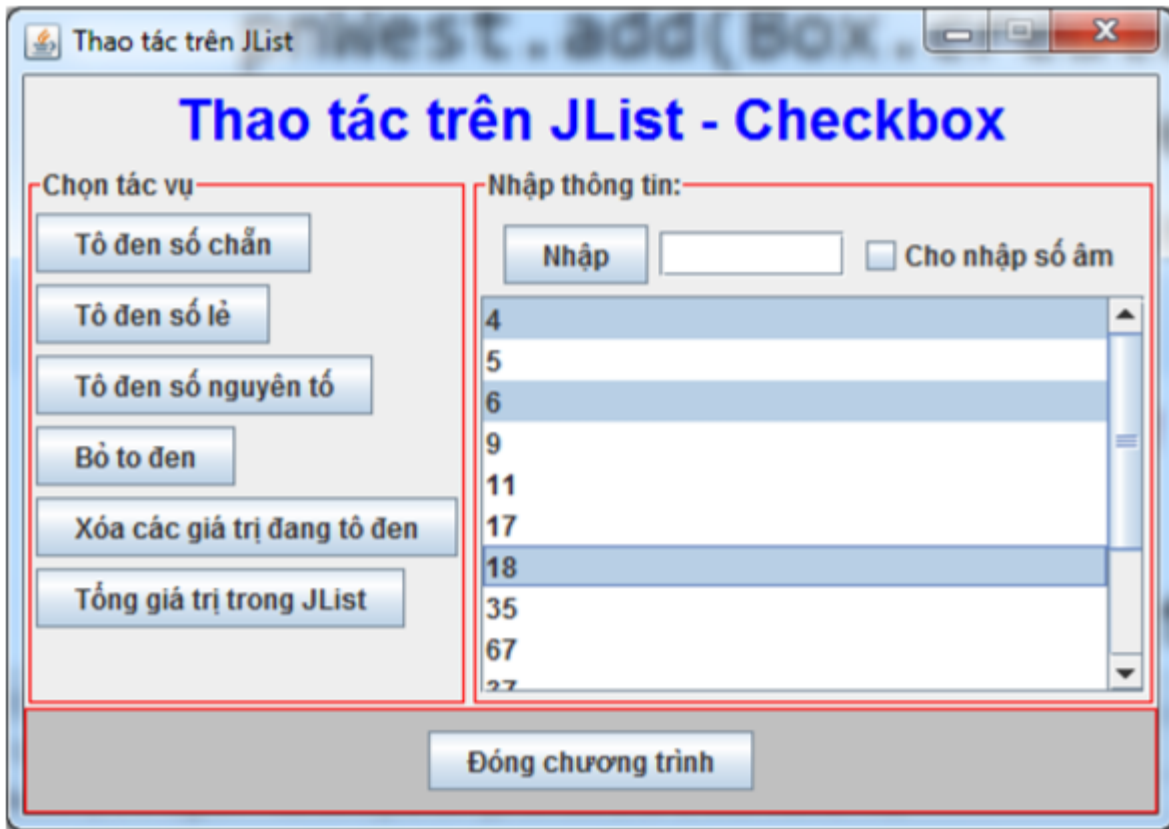
b) Các giá trị a và b bắt buộc nhập giá trị số. Nếu người dùng chưa nhập hoặc nhập không đúng, ứng dụng sẽ hiển thị thông báo nhắc.

c) Khi bấm nút Giải thì tùy thuộc vào phép toán được chọn mà thực hiện xử lý và thông báo lỗi.

d) Khi người dùng nhấn nút xóa thì thiết lập nội dung hiển thị của các thành phần trên màn hình giống như lúc ban đầu.

e) Khi người dùng nhấn nút Thoát thì chương trình sẽ hiển thị thông báo hỏi người dùng có muốn thoát hay không, nếu đồng ý thì cửa sổ chương trình sẽ đóng.

18. Xây dựng ứng dụng thao tác trên Jlist, JTextField, Jcheckbox



- a) Chương trình cho phép nhập vào các số nguyên trong phần nhập thông tin, Khi người nhập giá trị và click nút “Nhập” thì sẽ cập nhập dữ liệu xuống JList, Nếu checked vào “Cho nhập số âm” thì các số âm mới được đưa vào JList còn không thì thông báo lỗi.
- b) Thực hiện các yêu cầu tại khung “Chọn tác vụ”

19. Nút Đóng chương trình: Hiển thông báo hỏi người sử dụng có muốn đóng giao diện hay không. Nếu có thì tắt toàn bộ cửa sổ của chương trình.

Chương 5: LUỒNG VÀ TẬP TIN (STREAMS & FILES)

5.1. Mở đầu

Việc lưu trữ dữ liệu trong các biến chương trình, các mảng có tính chất tạm thời và dữ liệu sẽ mất đi khi biến ra khỏi tầm ảnh hưởng của nó hoặc khi chương trình kết thúc. Files giúp cho các chương trình có thể lưu trữ một lượng lớn dữ liệu, cũng như có thể lưu trữ dữ liệu trong một thời gian dài ngay cả khi chương trình kết thúc. Trong chương này chúng ta sẽ tìm hiểu làm thế nào các chương trình java có thể tạo, đọc, ghi và xử lý các files tuần tự và các file truy cập ngẫu nhiên thông qua một số ví dụ minh họa.

Xử lý files là một vấn đề hết sức cơ bản, quan trọng mà bất kỳ một ngôn ngữ lập trình nào cũng phải hỗ trợ những thư viện, hàm để xử lý một số thao tác cơ bản nhất đối với kiểu dữ liệu file.

Xử lý files là một phần của công việc xử lý các luồng, giúp cho một chương trình có thể đọc, ghi dữ liệu trong bộ nhớ, trên files và trao đổi dữ liệu thông qua các kết nối trên mạng.

Chương này sẽ cung cấp cho chúng ta những kiến thức cơ bản về luồng (streams) và files:

- Thư viện các lớp về luồng trong java: luồng byte, luồng ký tự.
- Xuất nhập Console dùng luồng byte, luồng ký tự.
- Xuất nhập files dùng luồng ký tự và luồng byte.
- Vấn đề xử lý files truy cập ngẫu nhiên dùng lớp RandomAccessFile.
- Xử lý file và thư mục dùng lớp File.

5.2. Luồng (Streams)

5.2.1. Khái niệm luồng

Tất cả những hoạt động nhập/xuất dữ liệu (nhập dữ liệu từ bàn phím, lấy dữ liệu từ mạng về, ghi dữ liệu ra đĩa, xuất dữ liệu ra màn hình, máy in, ...) đều được quy về một khái niệm gọi là luồng (stream). Luồng là nơi có thể “sản xuất” và “tiêu thụ” thông tin. Luồng thường được hệ thống xuất nhập trong java gắn kết với một thiết bị vật lý. Tất cả các luồng đều có chung một nguyên tắc hoạt động ngay cả khi chúng được gắn kết với các thiết bị vật lý khác nhau. Vì vậy cùng một lớp, phương thức xuất nhập có thể dùng chung cho các thiết bị vật lý khác nhau. Chẳng hạn cùng một phương thức có thể dùng để ghi dữ liệu ra console, đồng thời có thể dùng để ghi dữ liệu xuống một file trên đĩa.java hiện thực luồng bằng tập hợp các lớp phân cấp trong gói *java.io*.

Java định nghĩa hai kiểu luồng: byte và ký tự (phiên bản gốc chỉ định nghĩa kiểu luồng byte, và sau đó luồng ký tự được thêm vào trong các phiên bản về sau).

Luồng byte (hay luồng dựa trên byte) hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.

Luồng ký tự được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode). Trong một vài trường hợp luồng ký tự sử dụng hiệu quả hơn luồng byte, nhưng ở mức hệ thống thì tất cả những xuất nhập đều phải quy về byte. Luồng ký tự hỗ trợ hiệu quả chỉ đối với việc quản lý, xử lý các ký tự.

5.2.2. Luồng byte (Byte Streams)

Các luồng byte được định nghĩa dùng hai lớp phân cấp. Mức trên cùng là hai lớp trừu tượng *InputStream* và *OutputStream*. *InputStream* định nghĩa những đặc điểm chung cho những luồng nhập byte. *OutputStream* mô tả cách xử lý của các luồng xuất byte.

Các lớp con dẫn xuất từ hai lớp *InputStream* và

OutputStream sẽ hỗ trợ chi tiết tương ứng với việc đọc ghi dữ liệu trên những thiết bị khác nhau. Đừng choáng ngợp với hàng loạt rất nhiều các lớp khác nhau. Đừng quá lo lắng, mỗi khi bạn nắm vững, sử dụng thành thạo một luồng byte nào đó thì bạn dễ dàng làm việc với những luồng còn lại.

Lớp luồng byte	Ý nghĩa
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream đọc dữ liệu từ một mảng byte
ByteArrayOutputStream	Output stream ghi dữ liệu đến một mảng byte
DataInputStream	Luồng nhập có những phương thức đọc những kiểu dữ liệu chuẩn trong java
DataOutputStream	Luồng xuất có những phương thức ghi những kiểu dữ liệu chuẩn trong java
FileInputStream	Luồng nhập cho phép đọc dữ liệu từ file
FileOutputStream	Luồng xuất cho phép ghi dữ liệu xuống file
FilterInputStream	Hiện thực lớp trừu tượng <i>InputStream</i>
FilterOutputStream	Hiện thực lớp trừu tượng <i>OutputStream</i>
InputStream	lớp cha của tất cả các lớp luồng nhập kiểu Byte
OutputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp xuất nhập kiểu Byte
PipedInputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng xuất kiểu ống.
PipedOutputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng nhập kiểu ống để tạo nên một kết nối trao đổi dữ liệu kiểu ống.
PrintStream	Luồng xuất có chứa phương thức <i>print()</i> và <i>println()</i>

PushbackInputStream	Là một luồng nhập kiểu Byte mà hỗ trợ thao tác trả lại (push back) và phục hồi thao tác đọc một byte (unread)
RandomAccessFile	Hỗ trợ các thao tác đọc, ghi đối với file truy cập ngẫu nhiên.
SequenceInputStream	Là một luồng nhập được tạo nên bằng cách nối kết logic các luồng nhập khác.

5.2.3. Luồng ký tự (Character Streams)

Các luồng ký tự được định nghĩa dùng hai lớp phân cấp.

Mức trên cùng là hai lớp trừu tượng *Reader* và *Writer*. Lớp *Reader* dùng cho việc nhập dữ liệu của luồng, lớp *Writer* dùng cho việc xuất dữ liệu của luồng. Những lớp dẫn xuất từ *Reader* và *Writer* thao tác trên các luồng ký tự Unicode.

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

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

5. 2. 4. Những luồng được định nghĩa trước (The Predefined Streams)

Tất cả các chương trình viết bằng java luôn tự động import gói *java.lang*. Gói này có định nghĩa lớp *System*, bao gồm một số đặc điểm của môi trường run-time, nó có ba biến luồng được định nghĩa trước là *in*, *out* và *err*, các biến này là các fields được khai báo *static* trong lớp *System*.

- *System.out*: luồng xuất chuẩn, mặc định là console. *System.out* là một đối tượng kiểu *PrintStream*.
- *System.in*: luồng nhập chuẩn, mặc định là bàn phím. *System.in* là một đối tượng kiểu *InputStream*.
- *System.err*: luồng lỗi chuẩn, mặc định cũng là console. *System.out* cũng là một đối tượng kiểu *PrintStream* giống *System.out*.

5. 3. Sử dụng luồng Byte

Như chúng ta đã biết hai lớp *InputStream* và *OutputStream* là hai siêu lớp (cha) đối với tất cả những lớp luồng xuất nhập kiểu byte. Những phương thức trong hai siêu lớp này ném ra các lỗi kiểu *IOException*. Những phương thức định nghĩa trong hai siêu lớp này là có thể dùng trong các lớp con của chúng. Vì vậy tập các phương thức đó là tập tối thiểu các chức năng nhập xuất mà những luồng nhập xuất kiểu byte có thể sử dụng.

Những phương thức định nghĩa trong lớp

InputStream và *OutputStream*

Phương thức	Ý nghĩa
InputStream	
<i>int available()</i>	Trả về số lượng bytes có thể đọc được từ luồng nhập

<code>void close()</code>	Đóng luồng nhập và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi <code>IOException</code>
<code>void mark(int numBytes)</code>	Đánh dấu ở vị trí hiện tại trong luồng nhập
<code>boolean markSupported()</code>	Kiểm tra xem luồng nhập có hỗ trợ phương thức <code>mark()</code> và <code>reset()</code> không.
<code>int read()</code>	Đọc byte tiếp theo từ luồng nhập
<code>int read(byte buffer[])</code>	Đọc <code>buffer.length</code> bytes và lưu vào trong vùng nhớ <code>buffer</code> . Kết quả trả về số bytes thật sự đọc được
<code>int read(byte buffer[], int offset, int numBytes)</code>	Đọc <code>numBytes</code> bytes bắt đầu từ địa chỉ <code>offset</code> và lưu vào trong vùng nhớ <code>buffer</code> . Kết quả trả về số bytes thật sự đọc được
<code>void reset()</code>	Nhảy con trỏ đến vị trí được xác định bởi việc gọi hàm <code>mark()</code> lần sau cùng.
<code>long skip(long numBytes)</code>	Nhảy qua <code>numBytes</code> dữ liệu từ luồng nhập
OutputStream	
<code>void close()</code>	Đóng luồng xuất và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi <code>IOException</code>
<code>void flush()</code>	Ép dữ liệu từ bộ đệm phải ghi ngay xuống luồng (nếu có)
<code>void write(int b)</code>	Ghi byte dữ liệu chỉ định xuống luồng
<code>void write(byte buffer[])</code>	Ghi <code>buffer.length</code> bytes dữ liệu từ mảng chỉ định xuống luồng
<code>void write(byte buffer[], int offset, int numBytes)</code>	Ghi <code>numBytes</code> bytes dữ liệu từ vị trí <code>offset</code> của mảng chỉ định <code>buffer</code> xuống luồng

5.3.1. Đọc dữ liệu từ Console

Trước đây, khi Java mới ra đời để thực hiện việc nhập dữ liệu từ Console người ta chỉ dùng luồng nhập byte. Về sau thì chúng ta có thể dùng cả luồng byte và luồng ký tự, nhưng trong một số trường hợp thực tế để đọc dữ liệu từ Console người ta thích dùng luồng kiểu ký tự hơn, vì lý do đơn giản và dễ bảo trì chương trình. Ở đây với mục đích minh họa chúng ta dùng luồng byte thực hiện việc nhập xuất Console.

Ví dụ: chương trình minh họa việc đọc một mảng bytes từ `System.in`

```

import java.io.*;
class ReadBytes {
    public static void main(String args[]) throws IOException {
        byte data[] = new byte[100];
        System.out.print("Enter some characters. ");
    }
}

```

```
        System.in.read(data); System.out.print("You entered: ");
        for(int i=0; i < data.length; i++) System.out.print((char) data[i]);
    }
}
```

5.3.2. Xuất dữ liệu ra Console

Tương tự như nhập dữ liệu từ Console, với phiên bản đầu tiên của java để xuất dữ liệu ra Console ta chỉ có thể sử dụng luồng byte. Kể từ phiên bản 1.1 (có thêm luồng ký tự), để xuất dữ liệu ra Console có thể sử dụng cả luồng ký tự và luồng byte. Tuy nhiên, cho đến nay để xuất dữ liệu ra Console thường người ta vẫn dùng luồng byte.

Chúng ta đã khá quen thuộc với phương thức `print()` và `println()`, dùng để xuất dữ liệu ra Console. Bên cạnh đó chúng ta cũng có thể dùng phương thức `write()`.

Ví dụ: minh họa sử dụng phương thức `System.out.write()` để xuất ký tự 'X' ra Console

```
import java.io.*;
class WriteDemo {
    public static void main(String args[]) {
        int b;
        b = 'X';
        System.out.write(b);
        System.out.write('\n');
    }
}
```

5.3.3. Đọc và ghi file dùng luồng Byte

Tạo một luồng Byte gắn với file chỉ định dùng `FileInputStream` và `FileOutputStream`. Để mở một file, đơn giản chỉ cần tạo một đối tượng của những lớp này, tên file cần mở là thông số trong constructor. Khi file mở, việc đọc và ghi dữ liệu trên file được thực hiện một cách bình thường thông qua các phương thức cung cấp trong luồng.

5.3.3.1. Đọc dữ liệu từ file

- Mở một file để đọc dữ liệu

`FileInputStream(String fileName)` throws

`FileNotFoundException`

Nếu file không tồn tại: thì ném ra

`FileNotFoundException`

- Đọc dữ liệu: dùng phương thức *`read()` `int read()` throws `IOException`*: đọc từng byte từ file và trả về giá trị của byte đọc được. Trả về -1 khi hết file, và ném ra `IOException` khi có lỗi đọc.

- Đóng file: dùng phương thức *`close()` `void close()` throws `IOException`*: sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

Ví dụ: Hiển thị nội dung của một file tên `test.txt` lưu tại `D:\test.txt`

```
import java.io.*;
class ShowFile {
    public static void main(String args[]) throws IOException {
        int i;
        FileInputStream fin;
        try {fin = new FileInputStream("D:\\test.txt"); }
        catch(FileNotFoundException exc) {
            System.out.println("File Not Found");
            return;
        }
        catch(ArrayIndexOutOfBoundsException exc) {
            System.out.println("Usage: ShowFile File");
            return;
        }
        do {
            i = fin. read();
            if(i != -1) System.out.print((char) i);
        } while(i != -1); fin.close();
    }
}
```

5. 3. 3. 2 Ghi dữ liệu xuống file

- Mở một file để ghi dữ liệu

FileOutputStream(String fileName) throws

FileNotFoundException

Nếu file không tạo được: thì ném ra

FileNotFoundException

- Ghi dữ liệu xuống: dùng phương thức *write()* *void write(int byteval) throws IOException*: ghi một byte xác định bởi tham số *byteval* xuống file, và ném ra *IOException* khi có lỗi ghi.

- Đóng file: dùng phương thức *close()*

void close() throws IOException: sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

Ví dụ: copy nội dung một file text đến một file text khác.

```
import java.io.*;
class CopyFile {
    public static void main(String args[])    throws IOException {
        int i;
        FileInputStream fin;
        FileOutputStream fout;
        try {
            try{ fin = new FileInputStream("D:\\source.txt"); }

```

```

        catch(FileNotFoundException exc){
            System.out.println("Input File Not Found");
            return;
        }
        try { fout = new FileOutputStream("D:\\dest.txt"); }
        catch(FileNotFoundException exc) {
            System.out.println("Error Opening Output File");
            return;
        }
    }
    catch(ArrayIndexOutOfBoundsException exc) {
        System.out.println("Usage: CopyFile From To");
        return;
    }
}

try {
    do {
        i = fin. read();
        if(i != -1) fout. write(i);
    } while(i != -1);
}
catch(IOException exc) { System.out.println("File Error"); }
fin.close();  fout.close();
}
}

```

5.3.4. Đọc và ghi dữ liệu nhị phân

Phản trên chúng ta đã đọc và ghi các bytes dữ liệu là các ký tự mã ASCII. Để đọc và ghi những giá trị nhị phân của các kiểu dữ liệu trong java, chúng ta sử dụng *DataInputStream* và *DataOutputStream*.

DataOutputStream: hiện thực interface *DataOutput*.interface *DataOutput* có các phương thức cho phép ghi tất cả những kiểu dữ liệu cơ sở của java đến luồng (theo định dạng nhị phân).

Phương thức	Ý nghĩa
<i>void writeBoolean (boolean val)</i>	Ghi xuống luồng một giá trị boolean được xác định bởi val.
<i>void writeByte (int val)</i>	Ghi xuống luồng một byte được xác định bởi val.
<i>void writeChar (int val)</i>	Ghi xuống luồng một Char được xác định bởi val.
<i>void writeDouble (double val)</i>	Ghi xuống luồng một giá trị Double được xác định bởi val.
<i>void writeFloat (float val)</i>	Ghi xuống luồng một giá trị float được xác định bởi val.

<i>void writeInt (int val)</i>	Ghi xuống luồng một giá trị int được xác định bởi val.
<i>void writeLong (long val)</i>	Ghi xuống luồng một giá trị long được xác định bởi val.
<i>void writeShort (int val)</i>	Ghi xuống luồng một giá trị short được xác định bởi val.

Contructor: *DataOutputStream(OutputStream outputStream)*

OutputStream: là luồng xuất dữ liệu. Để ghi dữ liệu ra file thì đối tượng *outputStream* có thể là *FileOutputStream*.

DataInputStream: hiện thực interface *DataInput.interface DataInput* có các phương thức cho phép đọc tất cả những kiểu dữ liệu cơ sở của java (theo định dạng nhị phân).

Phương thức	Ý nghĩa
<i>boolean readBoolean()</i>	Đọc một giá trị boolean
<i>Byte readByte()</i>	Đọc một byte
<i>char readChar()</i>	Đọc một Char
<i>double readDouble()</i>	Đọc một giá trị Double
<i>float readFloat()</i>	Đọc một giá trị float
<i>int readInt()</i>	Đọc một giá trị int
<i>Long readLong()</i>	Đọc một giá trị long
<i>short readShort()</i>	Đọc một giá trị short

Contructor: *DataInputStream(InputStream inputStream)* *InputStream*: là luồng nhập dữ liệu. Để đọc dữ liệu từ file thì đối tượng *InputStream* có thể là *FileInputStream*.

Ví dụ: dùng *DataOutputStream*, *DataInputStream* để ghi và đọc những kiểu dữ liệu khác trên file.

```
import java.io.*;
```

```
class RWData {
    public static void main(String args[]) throws IOException {
        DataOutputStream dataOut;
        DataInputStream dataIn;
        int i = 10;
        double d = 1023. 56;
        boolean b = true;
        try {dataOut = new DataOutputStream(new FileOutputStream("D:\\testdata")); }
        catch(IOException exc) {
            System.out.println("Cannot open file. ");
            return;
        }
        try {
```



```
        System.out.println("Writing " + i);
        dataOut. writeInt(i);
        System.out.println("Writing " + d);
        dataOut. writeDouble(d);
        System.out.println("Writing " + b);
        dataOut. writeBoolean(b);
        System.out.println("Writing " + 12. 2 * 7. 4);
        dataOut. writeDouble(12. 2 * 7. 4);
    }
    catch(IOException exc){System.out.println("Write error. "); }
    dataOut.close();
    System.out.println();
    try {dataIn = new DataInputStream(new FileInputStream("D:\\testdata"));}
    catch(IOException exc) {
        System.out.println("Cannot open file. ");
        return;
    }
    try {
        i = dataIn. readInt();
        System.out.println("Reading " + i);
        d = dataIn. readDouble();
        System.out.println("Reading " + d);
        b = dataIn. readBoolean();
        System.out.println("Reading " + b);
        d = dataIn. readDouble();
        System.out.println("Reading " + d);
    }
    catch(IOException exc) { System.out.println("Read error. "); }
    dataIn.close();
}
}
```

5.4. File truy cập ngẫu nhiên (Random Access Files)

Bên cạnh việc xử lý xuất nhập trên file theo kiểu tuần tự thông qua các luồng, java cũng hỗ trợ truy cập ngẫu nhiên nội dung của một file nào đó dùng *RandomAccessFile*. *RandomAccessFile* không dẫn xuất từ *InputStream* hay

OutputStream mà nó hiện thực các interface *DataInput*, *DataOutput* (có định nghĩa các phương thức I/O cơ bản). *RandomAccessFile* hỗ trợ vấn đề định vị con trỏ file bên trong một file dùng phương thức *seek(long newPos)*.

Ví dụ: minh họa việc truy cập ngẫu nhiên trên file. Chương trình ghi 6 số kiểu double xuống file, rồi đọc lên theo thứ tự ngẫu nhiên.

```
import java.io.*;
class RandomAccessDemo {
```

```
public static void main(String args[]) throws IOException {
    double data[] = {19. 4, 10. 1, 123. 54, 33. 0, 87. 9, 74. 25};
    double d;
    RandomAccessFile raf;
    try { raf = new RandomAccessFile("D:\\random. dat", "rw"); }
    catch(FileNotFoundException exc) {
        System.out.println("Cannot open file. ");
        return ;
    }
    for(int i=0; i < data. length; i++) {
        try { raf. writeDouble(data[i]); }
        catch(IOException exc) {
            System.out.println("Error writing to file. ");
            return ;
        }
    }
    try {
        raf. seek(0);
        d = raf. readDouble();
        System.out.println("First value is " + d);
        raf. seek(8);
        d = raf. readDouble();
        System.out.println("Second value is " + d);
        raf. seek(8 * 3);
        d = raf. readDouble();
        System.out.println("Fourth value is " + d);
        System.out.println();
        System.out.println("Here is every other value: ");
        for(int i=0; i < data. length; i+=2) {
            raf. seek(8 * i);
            d = raf. readDouble();
            System.out.print(d + " ");
        }
        System.out.println("\n");
    }
    catch(IOException exc) { System.out.println("Error seeking or reading. "); }
    raf.close();
}
```

5.5. Sử dụng luồng ký tự

Chúng ta đã tìm hiểu và sử dụng luồng byte để xuất/nhập dữ liệu. Tuy có thể nhưng trong một số trường hợp luồng byte không phải là cách “lý tưởng” để quản lý xuất nhập dữ liệu kiểu character,

vì vậy java đã đưa ra kiểu luồng character phục vụ cho việc xuất nhập dữ liệu kiểu character trên luồng.

Mức trên cùng là hai lớp trừu tượng *Reader* và *Writer*. Lớp *Reader* dùng cho việc nhập dữ liệu của luồng, lớp *Writer* dùng cho việc xuất dữ liệu của luồng. Những lớp dẫn xuất từ *Reader* và *Writer* thao tác trên các luồng ký tự Unicode.

Những phương thức định nghĩa trong lớp trừu tượng *Reader* và *Writer*

Phương thức	Ý nghĩa
Reader	
<i>abstract void close()</i>	Đóng luồng
<i>void mark(int numChars)</i>	Đánh dấu vị trí hiện tại trên luồng
<i>boolean markSupported()</i>	Kiểm tra xem luồng có hỗ trợ thao tác đánh dấu <i>mark()</i> không?
<i>int read()</i>	Đọc một ký tự
<i>int read(char buffer[])</i>	Đọc buffer. length ký tự cho vào buffer
<i>abstract int read(char buffer[], int offset, int numChars)</i>	Đọc numChars ký tự cho vào vùng đệm buffer tại vị trí buffer[offset]
<i>boolean ready()</i>	Kiểm tra xem luồng có đọc được không?
<i>void reset()</i>	Dời con trỏ nhập đến vị trí đánh dấu trước đó
<i>long skip(long numChars)</i>	Bỏ qua numChars của luồng nhập
Writer	
<i>abstract void close()</i>	Đóng luồng xuất. Có lỗi ném ra IOException
<i>abstract void flush()</i>	Dọn dẹp luồng (buffer xuất)
<i>void write(int ch)</i>	Ghi một ký tự
<i>void write(byte buffer[])</i>	Ghi một mảng các ký tự
<i>abstract void write(char buffer[],int offset, int numChars)</i>	Ghi một phần của mảng ký tự
<i>void write(String str)</i>	Ghi một chuỗi
<i>void write(String str, int offset, int numChars)</i>	Ghi một phần của một chuỗi ký tự

5.5.1. Nhập Console dùng luồng ký tự

Thường thì việc nhập dữ liệu từ Console dùng luồng ký tự thì thuận lợi hơn dùng luồng byte. Lớp tốt nhất để đọc dữ liệu nhập từ Console là lớp *BufferedReader*. Tuy nhiên chúng ta không thể xây dựng một lớp *BufferedReader* trực tiếp từ *System.in*. Thay vào đó chúng ta phải chuyển nó thành một luồng ký tự. Để làm điều này chúng ta dùng *InputStreamReader* chuyển bytes thành ký tự.

Để có được một đối tượng *InputStreamReader* gắn với *System.in* ta dùng constructor của *InputStreamReader*. *InputStreamReader(InputStream inputStream)*

Tiếp theo dùng đối tượng *InputStreamReader* đã tạo ra để tạo ra một *BufferedReader* dùng constructor *BufferedReader*. *BufferedReader(Reader inputReader)*

Ví dụ: Tạo một *BufferedReader* gắn với Keyboard

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Sau khi thực hiện câu lệnh trên, *br* là một luồng ký tự gắn với Console thông qua *System.in*.

Ví dụ: Dùng *BufferedReader* đọc từng ký tự từ Console. Việc đọc kết thúc khi gặp dấu chấm (dấu chấm để kết thúc chương trình).

```
import java.io.*; class ReadChars {  
    public static void main(String args[]) throws IOException {  
        char c;  
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
  
        System.out.println("Nhap chuoi ky tu, gioi han dau cham. ");  
        do {  
            c = (char) br.read();  
            System.out.println(c);  
        } while(c != '.');  
    }  
}
```

5.5.2. Xuất Console dùng luồng ký tự

Trong ngôn ngữ java, bên cạnh việc dùng *System.out* để xuất dữ liệu ra Console (thường dùng để debug chương trình), chúng ta có thể dùng luồng *PrintWriter* đối với các chương trình “chuyên nghiệp”. *PrintWriter* là một trong những lớp luồng ký tự. Việc dùng các lớp luồng ký tự để xuất dữ liệu ra Console thường được “ưa chuộng” hơn.

Để xuất dữ liệu ra Console dùng *PrintWriter* cần thiết phải chỉ định *System.out* cho luồng xuất.

Ví dụ: Tạo đối tượng *PrintWriter* để xuất dữ liệu ra Console

```
PrintWriter pw = new PrintWriter(System.out, true);
```

Ví dụ: minh họa dùng *PrintWriter* để xuất dữ liệu ra Console

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

```
        PrintWriter pw = new PrintWriter(System.out, true);
        int i = 10;
        double d = 123. 67;
        double r = i+d;
        pw.println("Using a PrintWriter. ");
        pw.println(i);
        pw.println(d);
        pw.println(i + " + " + d + " = " + r);
    }
}
```

5.5.3. Đọc/ghi File dùng luồng ký tự

Thông thường để đọc/ghi file người ta thường dùng luồng byte, nhưng đối với luồng ký tự chúng ta cũng có thể thực hiện được. Ưu điểm của việc dùng luồng ký tự là chúng thao tác trực tiếp trên các ký tự Unicode. Vì vậy luồng ký tự là chọn lựa tốt nhất khi cần lưu những văn bản Unicode.

Hai lớp luồng thường dùng cho việc đọc/ghi dữ liệu ký tự xuống file là *FileReader* và *FileWriter*.

Ví dụ: Đọc những dòng văn bản nhập từ bàn phím và ghi chúng xuống file tên là “test.txt”. Việc đọc và ghi kết thúc khi người dùng nhập vào chuỗi “stop”.

```
import java.io.*;
class KtoD{
    public static void main(String args[]) throws IOException {
        String str;
        FileWriter fw;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        try { fw = new FileWriter("D:\\test.txt"); }
        catch(IOException exc) {
            System.out.println("Khong the mo file.");
            return ;
        }
        System.out.println("Nhap ('stop' de ket thuc chuong trinh). ");
        do {
            System.out.print(": ");
            str = br. readLine();
            if(str.compareTo("stop") == 0) break;
            str = str + "\r\n";
            fw. write(str);
        } while(str.compareTo("stop") != 0);
        fw.close();
    }
}
```

Ví dụ: đọc và hiển thị nội dung của file “test.txt” lên màn hình.

```
import java.io.*;
```

```
class DtoS {  
    public static void main(String args[]) throws Exception {  
        FileReader fr = new FileReader("D:\\test.txt");  
        BufferedReader br = new BufferedReader(fr);  
        String s;  
        while((s = br.readLine()) != null) { System.out.println(s); }  
        fr.close();  
    }  
}
```

5.6. Lớp File

Lớp File không phục vụ cho việc nhập/xuất dữ liệu trên luồng. Lớp File thường được dùng để biết được các thông tin chi tiết về tập tin cũng như thư mục (tên, ngày giờ tạo, kích thước, ...)

[java.lang. Object](#) +--[java. io. File](#)

Các Constructor:

Tạo đối tượng File từ đường dẫn tuyệt đối *public File([String](#) pathname)* ví dụ: *File f = new File("C:\\Java\\vd1.java");*

Tạo đối tượng File từ tên đường dẫn và tên tập tin tách biệt *public File([String](#) parent, [String](#) child)* ví dụ: *File f = new File("C:\\Java", "vd1.java");*

Tạo đối tượng File từ một đối tượng File khác *public File([File](#) parent, [String](#) child)*

ví dụ: *File dir = new File ("C:\\Java");*

File f = new File(dir, "vd1.java");

Một số phương thức thường gặp của lớp File (chi tiết về các phương thức đọc thêm trong tài liệu J2SE API Specification)

<i>public String getName()</i>	Lấy tên của đối tượng File
<i>public String getPath()</i>	Lấy đường dẫn của tập tin
<i>Public boolean isDirectory()</i>	Kiểm tra xem tập tin có phải là thư mục không?
<i>public boolean isFile()</i>	Kiểm tra xem tập tn có phải là một file không?
...	
<i>public String[] list()</i>	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.

Ví dụ:

```
import java. awt.*; import java.io.*;  
public class FileDemo {
```

```
public static void main(String args[]) {  
    Frame fr = new Frame ("File Demo");  
    fr.setBounds(10, 10, 300, 200);  
    fr.setLayout(new BorderLayout());  
    Panel p = new Panel(new GridLayout(1,2));  
    List list_C = new List();  
    list_C.add("C:\\");  
    File driver_C = new File ("C:\\");  
    String[] dirs_C = driver_C.list();  
    for (int i=0;i<dirs_C.length;i++) {  
        File f = new File ("C:\\\" + dirs_C[i]);  
        if (f.isDirectory()) list_C.add("<DIR>" + dirs_C[i]);  
        else list_C.add(" " + dirs_C[i]);  
    }  
    List list_D = new List();  
    list_D.add("D:\\");  
    File driver_D = new File ("D:\\");  
    String[] dirs_D = driver_D.list();  
    for (int i=0;i<dirs_D.length;i++) {  
        File f = new File ("D:\\\" + dirs_D[i]);  
        if (f.isDirectory()) list_D.add("<DIR>" + dirs_D[i]);  
        else list_D.add(" " + dirs_D[i]);  
    }  
    p.add(list_C);  
    p.add(list_D);  
    fr.add(p, BorderLayout.CENTER);  
    fr.setVisible(true);  
}
```

CÂU HỎI VÀ BÀI TẬP

1. Tạo file bt1.txt trong ổ D, gõ nội dung vào file và lưu lại. Viết chương trình hiển thị nội dung file bt1.txt.
2. Viết chương trình tạo file bt2.txt, sau đó ghi vào các ký tự từ a đến z.
3. Dùng DataOutputStream và DataInputStream để ghi dữ liệu kiểu int, double, boolean, double, char, string vào file bt3.txt. Đọc và hiển thị dữ liệu trong file bt3.txt
4. Viết chương trình tạo đối tượng SinhVien gồm các thuộc tính: họ tên, năm sinh, điểm trung bình. Ghi các thuộc tính này vào file theo kiểu nhị phân. Đọc và in thông tin của file này ra màn hình.
5. Viết chương trình nhập danh sách tên của 1 lớp học và ghi chúng xuống file tên là danhsach.txt. Việc đọc và ghi kết thúc khi người dùng nhập vào “stop”.

6. Viết chương trình đọc và hiển thị nội dung file danh sach.txt ở tập 5.
7. Thiết kế giao diện như sau:

Khi click vào nút lệnh LƯU thì toàn bộ nội dung trong TEXT sẽ được lưu vào tập tin d:/bai7.txt.

8. Thiết kế giao diện như sau:

Yêu cầu: khi click vào nút THÊM thì nội dung của TEXT sẽ được thêm vào cuối tập tin d:/bai8.txt

9. Thiết kế giao diện như sau:

Yêu cầu:

- Gõ đường dẫn đến tập tin đã có trên hệ thống.
- Khi click vào nút SHOW thì nội dung tập tin (vừa gõ trên ô Text) sẽ được hiển thị tại vị trí ô text nội dung.

10. Thực hiện các yêu cầu sau:

- Lớp CD nhằm đặc tả cho một đĩa CD. Các thuộc tính CD: Mã số CD (không cho phép sửa), tựa CD, ca sỹ, số bài hát và giá thành.
 - Tạo 2 constructors: Constructor có đầy đủ tham số và constructor có một tham số là mã số CD.
 - Viết các phương thức setters/getters.
 - Phương thức equals, so sánh bằng theo mã số CD.
- Lớp DanhSachCD
 - Dùng một ArrayList để quản lý danh sách CD.
 - Tạo 1 constructor không có tham số để khởi tạo danh sách
 - Viết phương thức thêm CD vào danh sách. Thêm thành công nếu không trùng mã số
 - Viết phương thức lấy thông tin CD tại vị trí i trong danh sách.
 - Viết phương thức xóa một CD khi biết mã số.
 - Viết phương thức tìm kiếm đĩa CD khi biết mã số.
 - Viết phương thức cập nhật thông tin cho đĩa CD khi biết mã số CD.
 - Viết phương thức trả về tổng số CD có trong danh sách
- Thiết kế giao diện như sau:

Yêu cầu xử lý:

- Nút “Thêm”, thêm một CD vào bảng với các cột như hình. Chú ý không được thêm khi:
 - Không nhập đủ dữ liệu.

- Số bài hát và giá không phải là số.
- Trùng mã CD.
- Khi chọn 1 dòng trên bảng thì hiện thông tin CD đó lên các ô nhập liệu.
- Nút “Clear”, xóa rỗng nội dung trong các JTextField, để người sử dụng nhập thông tin CD mới.
- Nút “Lưu”, lưu danh sách CD vào file. Lưu ý, khi chương trình bắt đầu thực hiện, nếu tồn tại file dữ liệu đã lưu trước đó, thì nạp danh sách CD lên trên bảng.
- Nút “Xóa”: Cho phép người sử dụng nhập mã CD cần xóa vào hộp thoại InputDialog. Nếu CD tồn tại, hỏi nhắc người sử dụng bằng hộp thoại ConfirmDialog trước khi xóa. Ngược lại, thông báo không tồn tại trong một hộp thoại MessageDialog.
- Nút “Tìm”: Cho phép người sử dụng nhập mã CD cần tìm vào hộp thoại InputDialog. Nếu CD tồn tại, thì hiển thị thông tin CD đó lên các ô nhập liệu. Ngược lại, thông báo không tồn tại trong một hộp thoại MessageDialog.
- Nút “Sửa”: Cho phép người sử dụng sửa thông tin CD. Lưu ý, không được sửa mã số CD.

Chương 6: LẬP TRÌNH CƠ SỞ DỮ LIỆU

6.1. GIỚI THIỆU

Hầu hết các chương trình máy tính hiện nay đều 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:

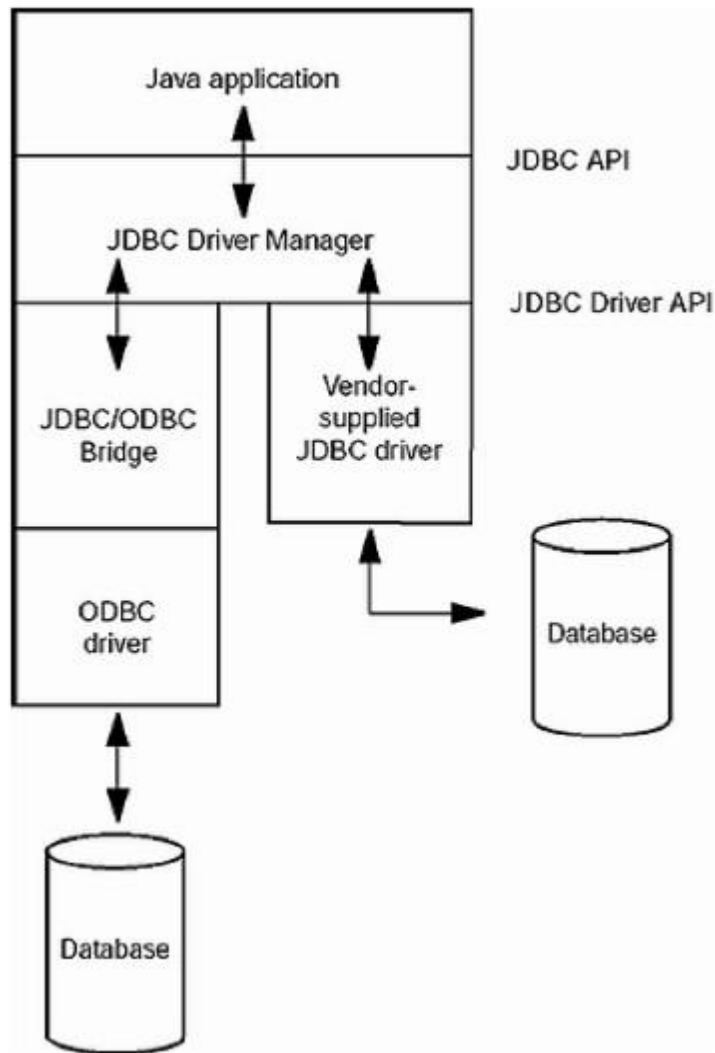
- java.sql: chứa các lớp và giao diện cơ sở của JDBC API.
- javax.sql: chứa các lớp và giao diện mở rộng.

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.

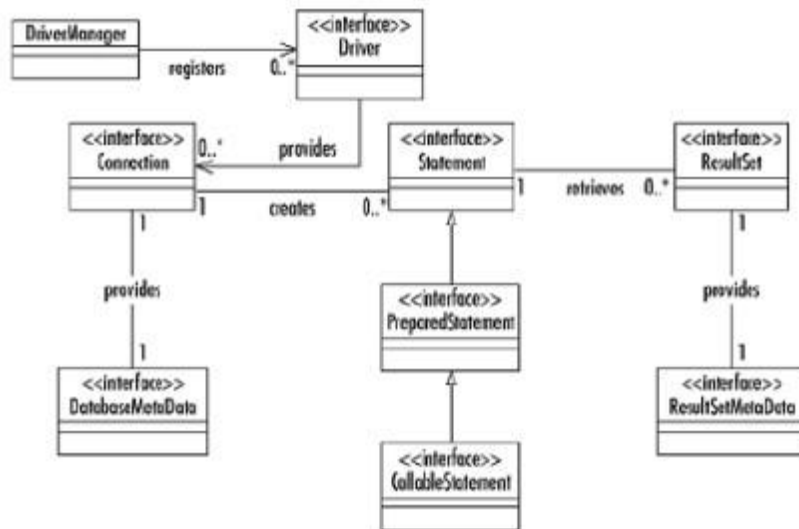
6.2. KIẾN TRÚC JDBC

Kiến trúc của của JDBC tương tự như kiến trúc ODBC do Microsoft xây dựng. Theo kiến trúc này các thao tác liên quan đến cơ sở dữ liệu trong chương trình được thực hiện thông qua các JDBC API. Sau đó các JDBC API sẽ truyền các yêu cầu của chương trình đến bộ quản lý trình điều khiển JDBC, là bộ phận có nhiệm vụ lựa chọn trình điều khiển thích hợp để có thể làm việc với cơ sở dữ liệu cụ thể mà chương trình muốn kết nối.

Như vậy kiến trúc của JDBC gồm 2 tầng: tầng đầu tiên là các JDBC API, có nhiệm vụ chuyển các câu lệnh SQL cho bộ quản lý trình điều khiển JDBC; tầng thứ 2 là các JDBC Driver API, thực hiện nhiệm vụ liên hệ với trình điều khiển của hệ quản trị cơ sở dữ liệu cụ thể.



Hình bên dưới minh họa các lớp và giao diện cơ bản trong JDBC API.



6.3. Các khái niệm cơ bản

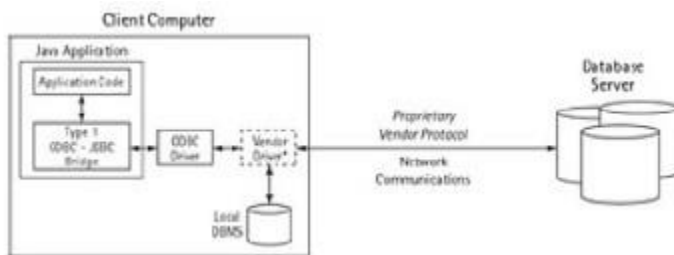
6.3.1. jdbc Driver

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.

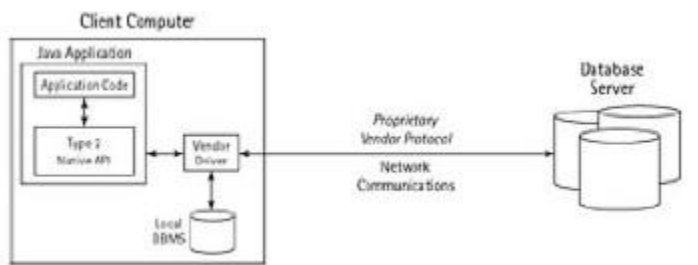
Danh sách các trình điều khiển JDBC cho các hệ quản trị CSDL khác nhau được Sun cung cấp và cập nhật liên tục tại địa chỉ:

<http://industry.java.sun.com/products/jdbc/drivers>. Các trình điều khiển JDBC được phân làm 04 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 trình 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`.

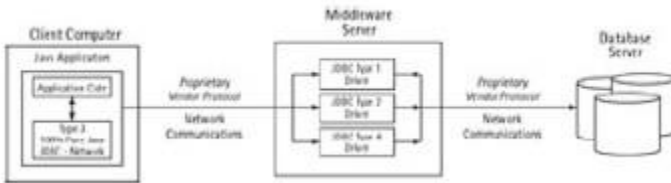


- Loại 2: có tên gọi là **Native API Driver**. Trình điều khiển loại này sẽ chuyển các lời gọi của JDBC API sang thư viện hàm (API) tương ứng với từng hệ CSDL cụ thể. Trình điều khiển loại này thường chỉ do nhà xây dựng hệ CSDL cung cấp. Để có thể thi hành chương trình mã lệnh để làm việc với hệ CSDL cụ thể cần phải được cung cấp đi kèm với chương trình.



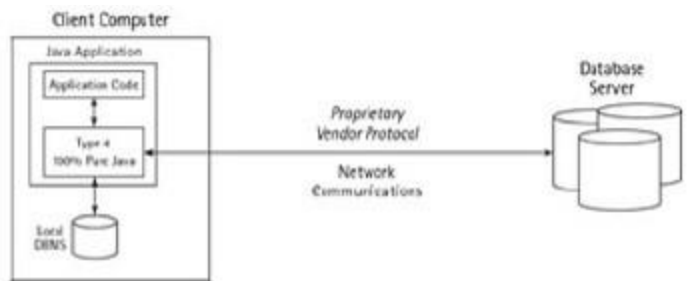
- Loại 3: có tên gọi là **JDBC-Net Driver**. Trình điều khiển loại này sẽ chuyển các lời gọi JDBC API sang một dạng chuẩn độc lập với các hệ

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.



- 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 thân Java, có nghĩa là không cần phải có mã lệnh của hệ CSDL cụ thể khi thi hành chương trình.



6.3.2.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.

6.4. 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`.

6.4.1. 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")`.

6.4.2. 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`:

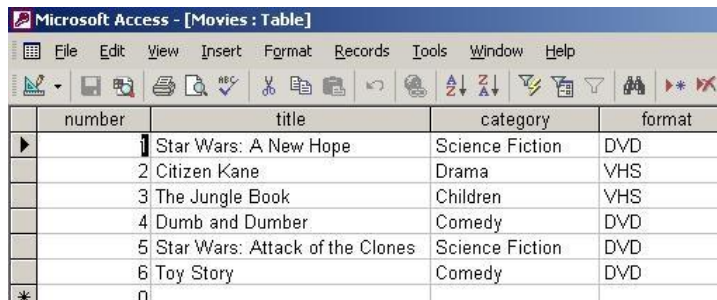
- `public static Connection getConnection(String url)` throws `SQLException`: thực hiện kết nối với CSDL được yêu cầu. Bộ quản lý trình điều khiển sẽ tự động lựa chọn trình điều khiển phù hợp trong số các trình điều khiển đã được nạp.
- `public static Connection getConnection(String url, String user, 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 đến CSDL.

6.4.3. Ví dụ

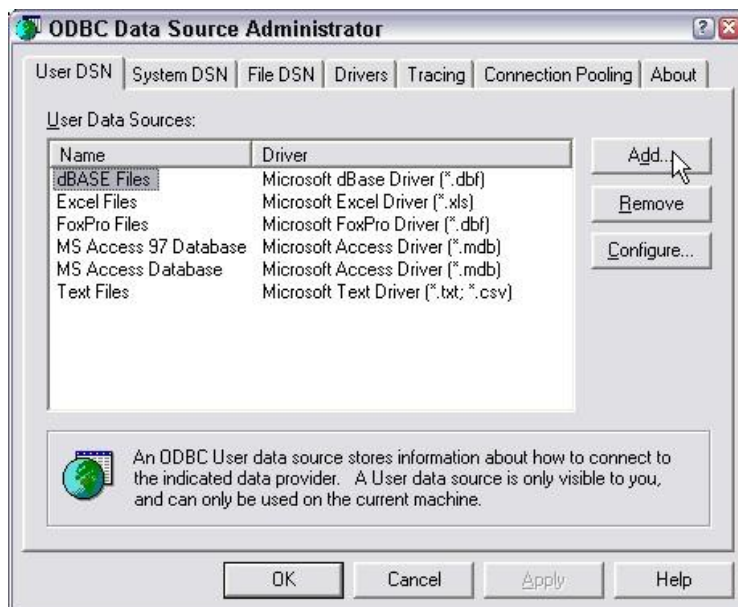
Trong phần ví dụ này chúng ta sẽ tìm hiểu các cách khác nhau để kết nối với tập tin CSDL Access movies.mdb có một bảng tên Movies. Bảng này gồm các cột number, title, category và format.

Để có thể tiến hành kết nối với Microsoft Access thông qua cầu



number	title	category	format
1	Star Wars: A New Hope	Science Fiction	DVD
2	Citizen Kane	Drama	VHS
3	The Jungle Book	Children	VHS
4	Dumb and Dumber	Comedy	DVD
5	Star Wars: Attack of the Clones	Science Fiction	DVD
6	Toy Story	Comedy	DVD

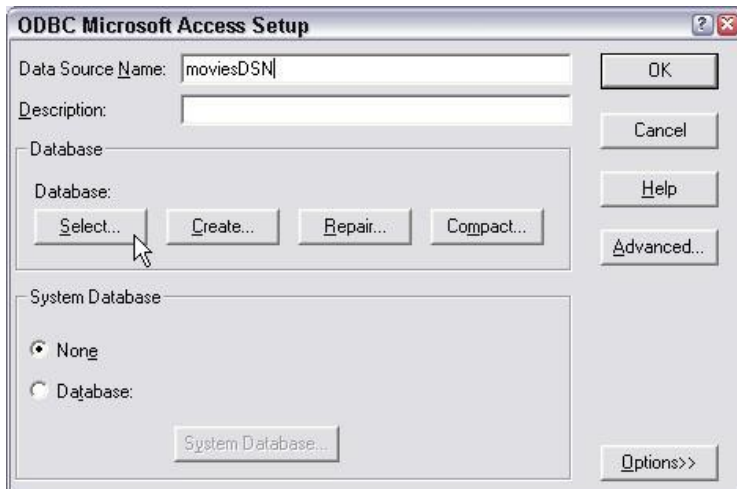
nối ODBC sau khi đã tạo tập tin CSDL movies.mdb, chúng ta cần phải tạo Data Source Name cho CSDL bằng cách vào Control Panel và chọn ODBC Data Source.



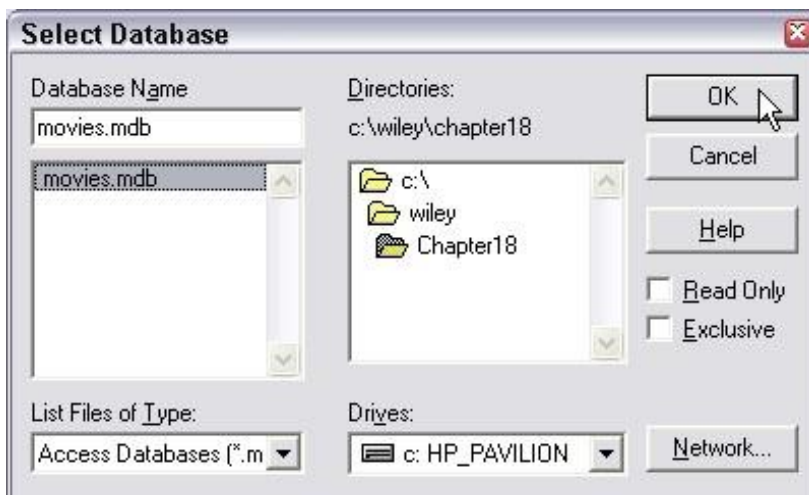
Tiếp theo nhấn vào nút Add, bạn sẽ thấy hiển thị danh sách các trình điều khiển CSDL hiện có.



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 {
        Class.forName("sun.jdbc.odbc.jdbcOdbcDriver");
        System.out.println("Loading the driver. . . ");
    }
    catch( Exception e ) {
        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!");
    }
    catch( SQLException e ) { e.printStackTrace( ); }
    finally {
        if( connection != null ) {
            try { connection.close( ); }
            catch( SQLException e ) { e.printStackTrace( ); }
        }
    }
}
```

```
}  
}  
}
```

6. 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 lại. 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[]
<i>BLOB</i>	java.sql.Blob
<i>CLOB</i>	Java.sql.Clob
<i>ARRAY</i>	Java.sql.Array
<i>REF</i>	Java.sql.Ref
<i>STRUCT</i>	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

6. 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 lệnh 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 một số nguyên. Các đối tượng của lớp ResultSet cho phép chúng ta truy cập đến kết quả trả về của các câu truy vấn.

6. 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 lệnh 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 nhiên phương thức này chỉ sử dụng được đối với các câu 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 cột cần 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 false.

6. 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(); }
}
}
```

6. 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 the Clones", "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(); }  
}
```

CÂU HỎI VÀ BÀI TẬP

1. Viết chương trình tạo, chọn, xóa cơ sở dữ liệu NhanVien trong JDBC
2. Viết chương trình tạo, xóa, chèn bản ghi vào cơ sở dữ liệu NhanVien trong JDBC

Tài liệu tham khảo:

- [1] java.sun.com
- [2] Herbert Schildt, **java 2. A Beginner's Guide**. Second Edition. McGraw-Hill - 2003.
- [3] Dr. Harvey M. Deitel - Paul J. Deitel, **java How to Program**, 4th Ed (Deitel). Prentice Hall - 2002
- [4] Simon Roberts – Philip Heller – Michael Ernest, **complete Java 2 Certification – study guide**. BPB Publications – 2000.
- [5] Cay S. Horstmann – Gary Cornell. **Core Java Volum 1 - Fundamentals**. The Sun Microsystems press. 1997
- [6] Cay S. Horstmann – Gary Cornell. **Core Java Volum 2 – Advanced Features**. The Sun Microsystems press. 1997

