



TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG

KHOA CÔNG NGHỆ THÔNG TIN

Chương 1

Căn bản về Ngôn ngữ lập trình Java

Số tiết: 15 tiết



The top 10 programming languages

❑ The TIOBE:

Jan 2020	Jan 2019	Change	Programming Language	Ratings	Change
1	1		Java	16.896%	-0.01%
2	2		C	15.773%	+2.44%
3	3		Python	9.704%	+1.41%
4	4		C++	5.574%	-2.58%
5	7	▲	C#	5.349%	+2.07%
6	5	▼	Visual Basic .NET	5.287%	-1.17%
7	6	▼	JavaScript	2.451%	-0.85%
8	8		PHP	2.405%	-0.28%
9	15	▲	Swift	1.795%	+0.61%
10	9	▼	SQL	1.504%	-0.77%



The top 10 programming languages

❑ The PYPL:

Worldwide, Jan 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.72 %	+4.3 %
2		Java	19.03 %	-1.9 %
3		Javascript	8.2 %	+0.1 %
4		C#	7.28 %	-0.2 %
5		PHP	6.09 %	-1.1 %
6		C/C++	5.91 %	-0.3 %
7		R	3.72 %	-0.2 %
8		Objective-C	2.47 %	-0.6 %
9		Swift	2.36 %	-0.2 %
10		Matlab	1.79 %	-0.2 %



Nội dung

1. Tổng quan về NNLT Java
2. Các thành phần cơ bản của NNLT Java
3. Các cấu trúc điều khiển
4. Mảng và chuỗi
5. Nhập xuất dữ liệu đơn giản
6. Ngoại lệ (Exception) và xử lý ngoại lệ
7. Lập trình hướng đối tượng trong Java



1. Tổng quan về NNLT Java

1.1. Lịch sử phát triển

1.2. Đặc điểm

1.3. Các công nghệ Java

1.4. Môi trường lập trình Java

1.5. Chương trình Java đơn giản

1.6. Quá trình phát triển một chương trình Java

1.7. IDE hỗ trợ lập trình Java



1.1. Lịch sử phát triển

- ❑ Tháng 4/1991 hãng Sun Microsystems thực hiện đề án với tên gọi là Green.
- ❑ Đề án Green nhằm đưa các kỹ thuật tin học vào những thiết bị điện tử thương mại như một cuộc đột phá về kỹ thuật, qua đó duy trì tốc độ phát triển cũng như lợi nhuận.
- ❑ Lãnh đạo đề án Green là James Gosling quyết định cần phải có một ngôn ngữ lập trình mới thỏa tính uyển chuyển (portability)



1.1. Lịch sử phát triển

- ❑ Tính uyển chuyển phải thỏa mãn không những về phía mã nguồn chương trình mà còn cả với bộ xử lý.
- ❑ Ngôn ngữ C++ tỏ ra khá uyển chuyển nhưng vẫn chưa đáp ứng được yêu cầu. Nếu sử dụng ngôn ngữ C++ thì khi gấp một bộ xử lý mới cần phải thay đổi chương trình dịch, rồi biên dịch lại chương trình.
- ❑ Tháng 8/1991, J.Gosling thiết kế ngôn ngữ mới và đặt tên là Oak (cây sồi).



1.1. Lịch sử phát triển

- ❑ Tháng 01/1995, Oak được đổi tên thành Java, tên một xứ sở ở Indonesia có giống cà phê nổi tiếng mà những người trong Green Team thường hay uống. Hơn nữa, Oak cũng trùng với tên một sản phẩm thương mại đã đăng ký.
- ❑ Ngày **23/05/1995**, Java được công bố chính thức ở Sun World'95



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Java được mô hình hóa trên nền C++ nhưng được đơn giản hóa và cải tiến.
- Một số người xem Java như là C++ bởi vì cú pháp nó cũng khá giống C++ nhưng nhiều chức năng hơn và ít khía cạnh tiêu cực hơn khi không hỗ trợ đa kế thừa, nạp chồng toán tử và con trỏ như C++.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Java được thiết kế ngay từ đầu theo hướng đối tượng trong khi các ngôn ngữ khác hỗ trợ cả hướng đối tượng và hướng thủ tục
- Hiện nay OOP là một cách tiếp cận phổ biến để xây dựng chương trình thay thế cho hướng thủ tục
- Một trong những vấn đề trung tâm trong phát triển phần mềm là làm thế nào để tái sử dụng mã.
- Lập trình hướng đối tượng cung cấp tính linh hoạt, mô-đun, rõ ràng và khả năng tái sử dụng tuyệt vời thông qua đóng gói, kế thừa và đa hình.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Điện toán phân tán liên quan đến một số máy tính làm việc cùng nhau trên một mạng.
- Java được thiết kế để làm cho tính toán phân tán dễ dàng. Vì khả năng kết nối mạng vốn đã được tích hợp vào Java, việc viết các chương trình mạng giống như gửi và nhận dữ liệu đến và đi từ một tệp.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Bạn cần một trình thông dịch để chạy các chương trình Java.
- Các chương trình được biên dịch thành mã Máy ảo Java gọi là bytecode.
- Bytecode độc lập với máy và có thể chạy trên bất kỳ máy nào có trình thông dịch Java, là một phần của Máy ảo Java (JVM).



Đặc điểm của Java

- ❑ Java Is Simple
- ❑ Java Is Object-Oriented
- ❑ Java Is Distributed
- ❑ Java Is Interpreted
- ❑ Java Is Robust
- ❑ Java Is Secure
- ❑ Java Is Architecture-Neutral
- ❑ Java Is Portable
- ❑ Java's Performance
- ❑ Java Is Multithreaded
- ❑ Java Is Dynamic

- Trình biên dịch Java có thể phát hiện nhiều vấn đề đầu tiên sẽ hiển thị tại thời điểm thực thi bằng các ngôn ngữ khác.
- Java đã loại bỏ một số loại cấu trúc lập trình dễ bị lỗi được tìm thấy trong các ngôn ngữ khác.
- Java có tính năng xử lý ngoại lệ trong theo thời gian thực để cung cấp hỗ trợ lập trình cho sự mạnh mẽ.



Đặc điểm của Java

- ❑ Java Is Simple
- ❑ Java Is Object-Oriented
- ❑ Java Is Distributed
- ❑ Java Is Interpreted
- ❑ Java Is Robust
- ❑ Java Is Secure
- ❑ Java Is Architecture-Neutral
- ❑ Java Is Portable
- ❑ Java's Performance
- ❑ Java Is Multithreaded
- ❑ Java Is Dynamic

➤ Java thực hiện một số cơ chế bảo mật để bảo vệ hệ thống của bạn chống lại tác hại gây ra bởi một số chương trình



Đặc điểm của Java

1. Java Is Simple
 2. Java Is Object-Oriented
 3. Java Is Distributed
 4. Java Is Interpreted
 5. Java Is Robust
 6. Java Is Secure
 7. Java Is Architecture-Neutral
 8. Java Is Portable
 9. Java's Performance
 10. Java Is Multithreaded
 11. Java Is Dynamic
- Write once, run anywhere
➤ Write one program that will run on any platform.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Bởi vì Java là kiến trúc trung lập, các chương trình Java là di động.
- Chúng có thể được chạy trên bất kỳ nền tảng nào mà không cần biên dịch lại.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

- Hiệu suất của Java tốt hơn vì Java là kiến trúc trung lập, các chương trình Java là di động.
- Chúng có thể được chạy trên bất kỳ nền tảng nào mà không cần biên dịch lại.



Đặc điểm của Java

1. Java Is Simple
2. Java Is Object-Oriented
3. Java Is Distributed
4. Java Is Interpreted
5. Java Is Robust
6. Java Is Secure
7. Java Is Architecture-Neutral
8. Java Is Portable
9. Java's Performance
10. Java Is Multithreaded
11. Java Is Dynamic

➤ Multithread programming is smoothly integrated in Java, whereas in other languages you have to call procedures specific to the operating system to enable multithreading.



Đặc điểm của Java

- ❑ Java Is Simple
- ❑ Java Is Object-Oriented
- ❑ Java Is Distributed
- ❑ Java Is Interpreted
- ❑ Java Is Robust
- ❑ Java Is Secure
- ❑ Java Is Architecture-Neutral
- ❑ Java Is Portable
- ❑ Java's Performance
- ❑ Java Is Multithreaded
- ❑ Java Is Dynamic

- Java được thiết kế để thích ứng với môi trường phát triển.
- Mã mới có thể được tải nhanh chóng mà không cần biên dịch lại.
- Không cần các nhà phát triển tạo và cho người dùng cài đặt các phiên bản phần mềm mới chính.
- Các tính năng mới có thể được kết hợp minh bạch khi cần thiết.



1.2. Tóm tắt đặc điểm JAVA

- ❑ Máy ảo Java (Java Virtual Machine)
- ❑ Vừa biên dịch và thông dịch
- ❑ Độc lập nền tảng
- ❑ Hướng đối tượng
- ❑ Đa luồng
- ❑ Khả chuyen



1.2. Đặc điểm

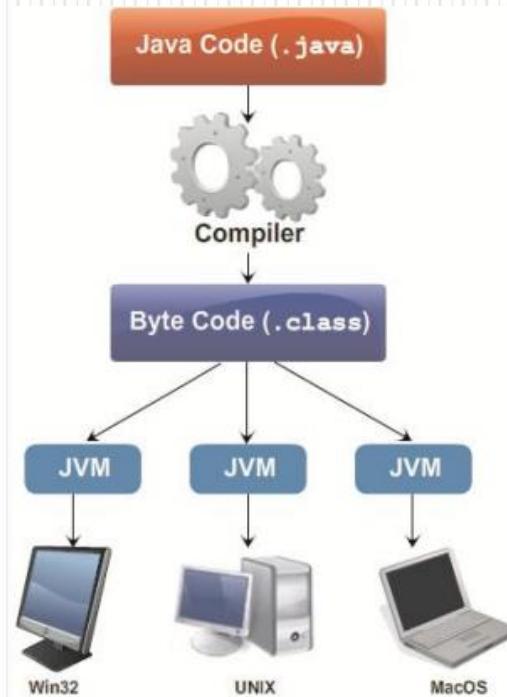
❑ Máy ảo Java (Java Virtual Machine)

- ❑ Tất cả các chương trình muốn thực thi được phải được biên dịch ra mã máy.
- ❑ 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 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 và trên các kiến trúc CPU khác nhau.



1.2. Đặc điểm

☐ Máy ảo Java (Java Virtual Machine)





1.2. Đặc điểm

❑ Vừa biên dịch vừa 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
- ❑ Sau đó file *.class sẽ được trình thông dịch thông dịch thành mã máy.



1.2. Đặc điểm

❑ Độc lập nền tảng

❑ 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).



1.2. Đặc điểm

❑ Hướng đối tượng

- ❑ Java là một ngôn ngữ hướng đối tượng hoàn toàn.
- ❑ Tất cả mọi thứ đề cập trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính của một chương trình viết bằng Java (hàm **main**) cũng phải đặt bên trong một lớp.



1.2. Đặc điểm

❑ Đa luồng

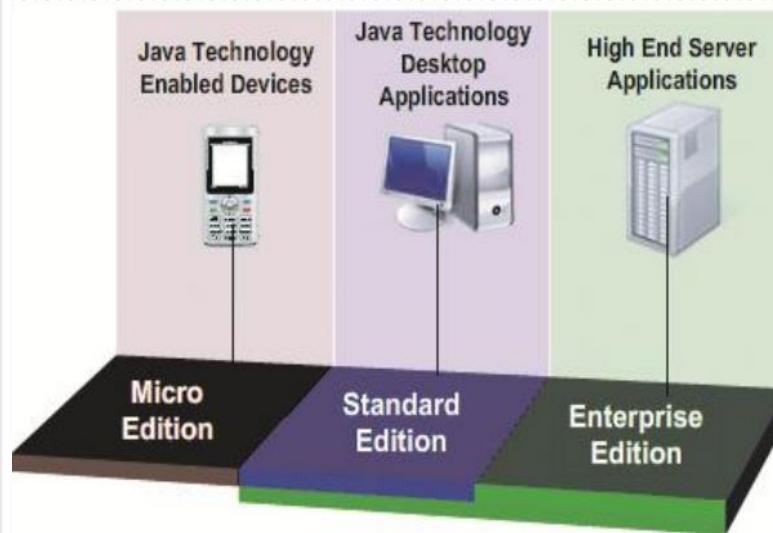
❑ Java hỗ trợ lập trình đa luồng cho phép nhiều tiến trình, tiểu trình có thể chạy song song cùng một thời điểm và tương tác với nhau.

❑ Hỗ trợ mạnh cho việc phát triển ứng dụng

❑ Công nghệ Java phát triển mạnh mẽ nhờ vào Sun Microsystems cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại ứng dụng khác nhau như: J2SE, J2EE J2ME.

1.3. Các công nghệ Java

- ❖ J2SE (Java 2 Platform Standard Edition)
- ❖ J2EE (Java 2 Platform Enterprise Edition)
- ❖ J2ME (Java 2 Platform Micro Edition)





1.3. Các công nghệ Java

❖ J2SE (Java 2 Platform Standard Edition)

- ❑ Là một nền tảng thực thi (bao gồm cả phát triển và triển khai) cho các ứng dụng Java. Nó cung cấp các API, các kiến trúc chuẩn, các thư viện lớp và các công cụ cốt lõi nhất để xây các ứng dụng Java.
- ❑ Là phiên bản chuẩn, là nền tảng thiên về phát triển các sản phẩm chạy trên máy tính để bàn (desktop-based), các ứng dụng dạng khách chủ (client server – based).



1.3. Các công nghệ Java

❑ J2EE (Java 2 Platform Enterprise Edition)

❑ Định nghĩa một chuẩn để phát triển những ứng dụng thương mại đa tầng (multitier enterprise applications). Các công nghệ trong J2EE bao gồm:

- ❑ Java Database Connectivity (JDBC) API
- ❑ Remote Method Invocation (RMI)
- ❑ Servlets và Java Server Pages (JSP)
- ❑ Java Transaction API (JTA)...



1.3. Các công nghệ Java

- ❑ **J2ME (Java 2 Platform Micro Edition)**
 - ❑ Là một nhánh của NNLT Java để phát triển các ứng dụng trên điện thoại di động, thiết bị cầm tay nhỏ gọn khác hay các thiết bị tiêu dùng với công suất hạn chế (TV, máy in, thiết bị điện tử,...).
 - ❑ Được phát triển trên phiên bản J2SE, nhưng nó lược bỏ đi khá nhiều các framework cũng như API để phù hợp với bộ nhớ cũng như tốc độ xử lý của các thiết bị di động.

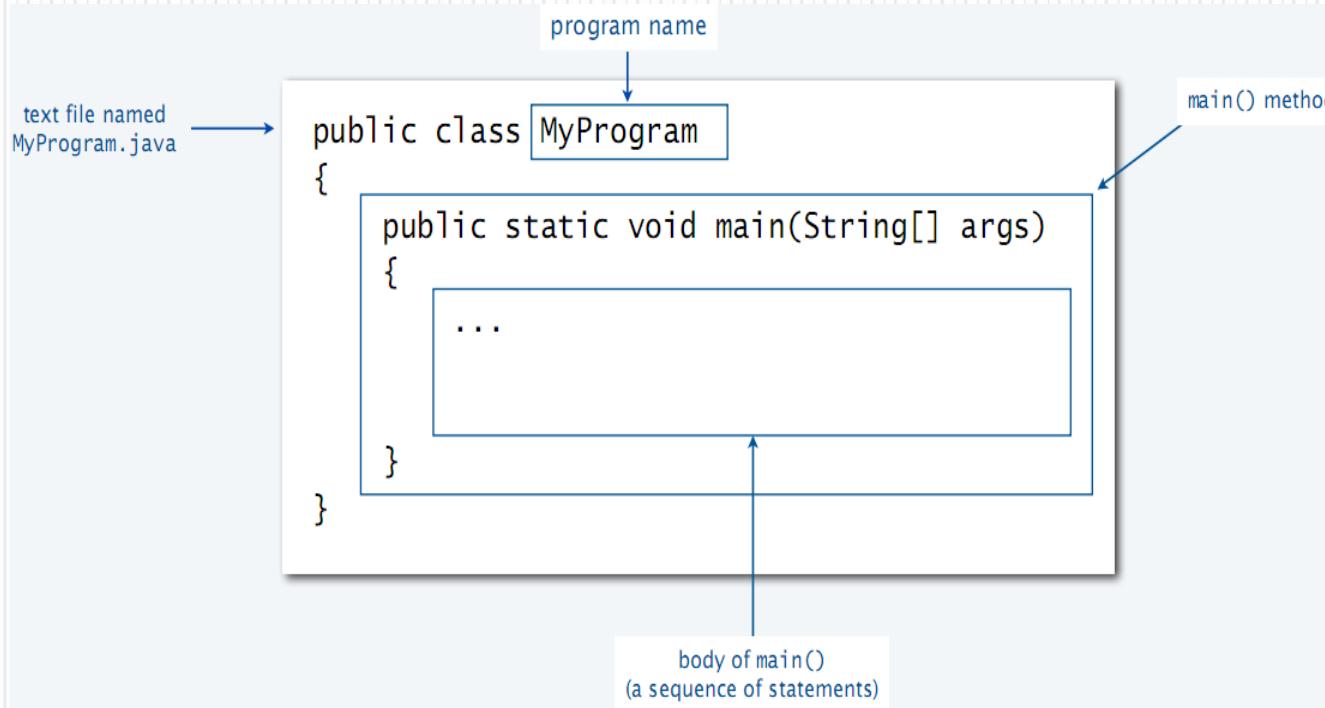


1.4. Môi trường lập trình Java

- ❑ Hãng SUN Microsystems cung cấp bộ công cụ **JDK** (Java Development Kit) và **JRE** (Java Runtime Environment) giúp thực thi chương trình Java.
- ❑ JDK còn được gọi là SDK (Java SDK – Software Development Kit) là bộ công cụ phát triển ứng dụng Java bao gồm 4 thành phần: Classes, Compiler, Debugger, Java Runtime Environment.

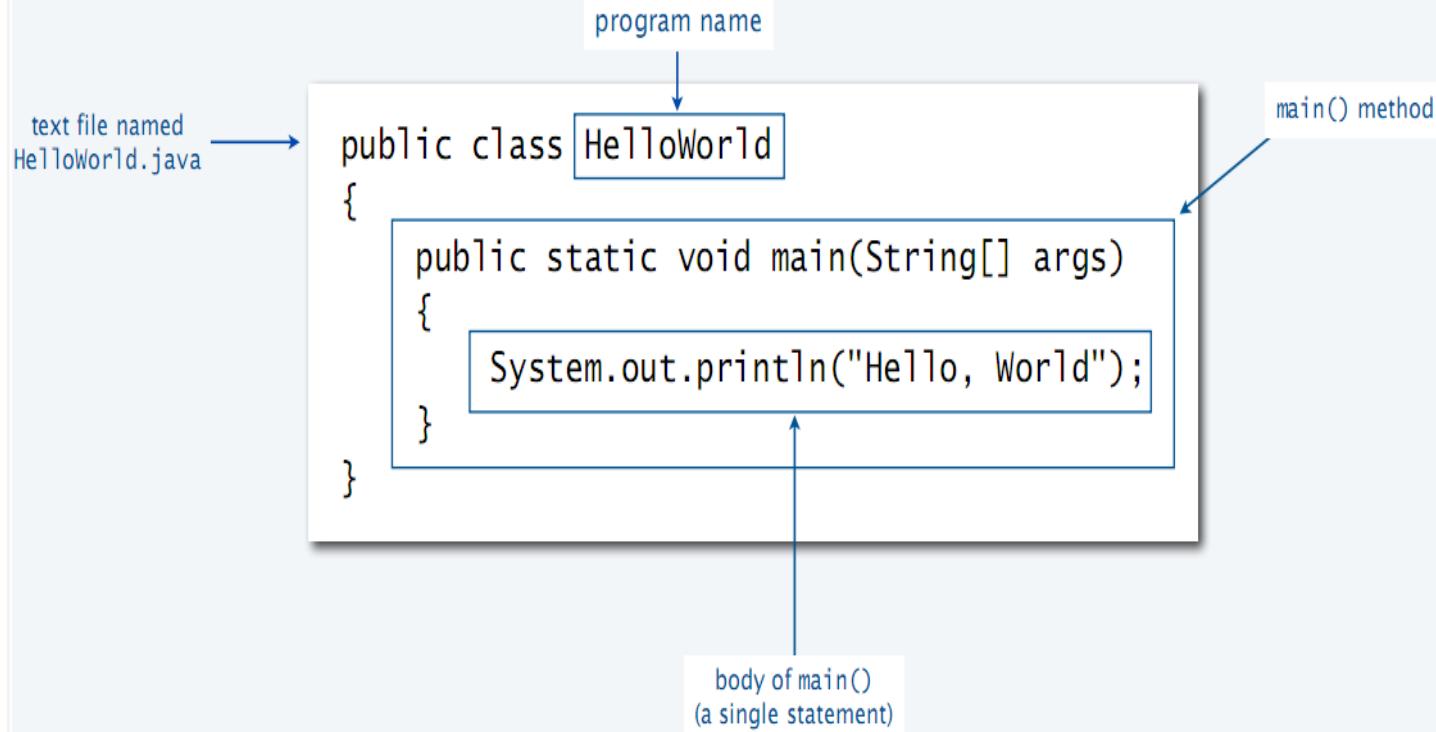


1.5. Chương trình Java đơn giản





1.5. Chương trình Java đơn giản





1.5. Chương trình Java đơn giản

B1. Mở Notepad để soạn thảo chương trình:

```
1  public class MyFirstJavaProgram {  
2  
3      /* This is my first java program.  
4          * This will print 'Hello World' as the output  
5          */  
6  
7      public static void main(String []args) {  
8          System.out.println("Hello World"); // prints Hello World  
9      }  
10 }
```

B2. Lưu lại với tập tin: MyFirstJavaProgram.java

B3. Mở command prompt: di chuyển đến thư mục đã lưu.

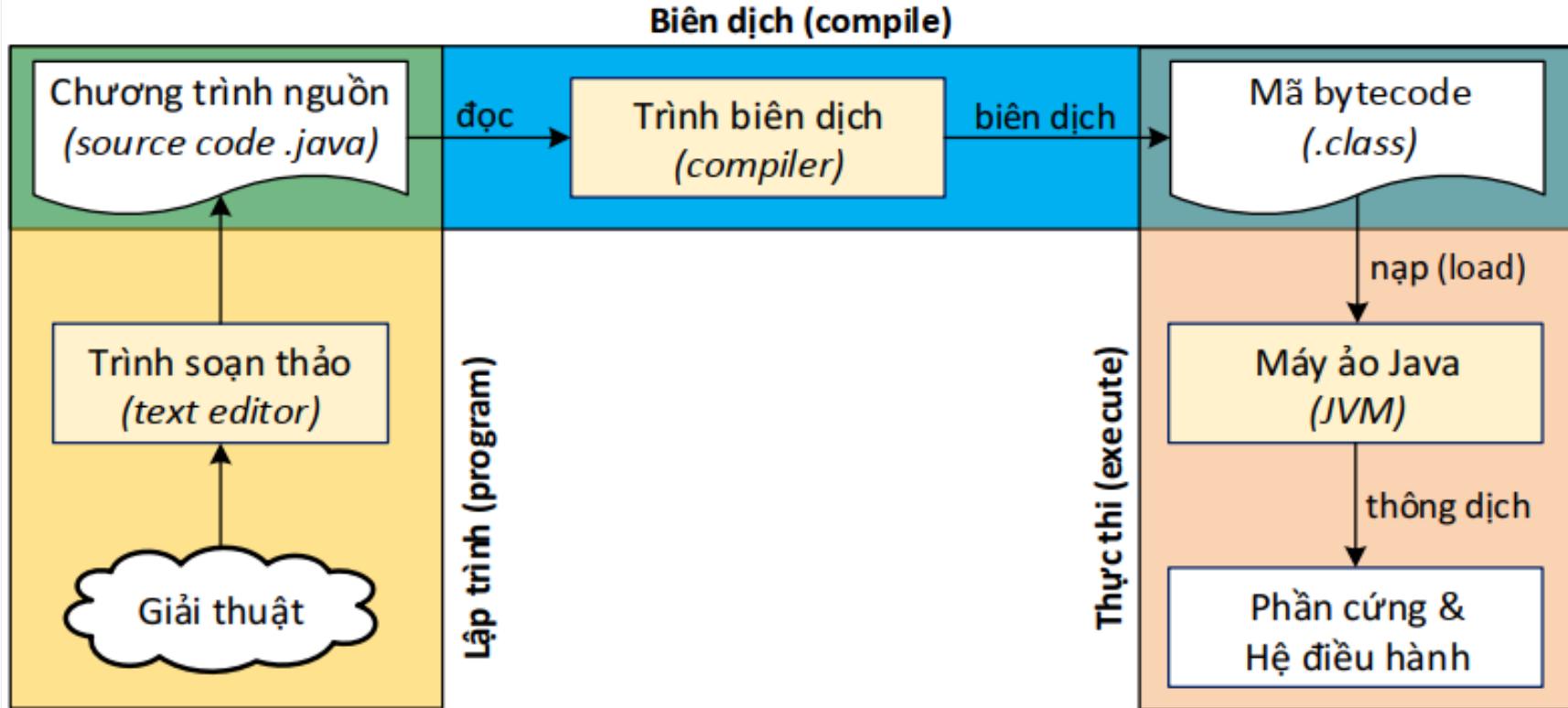
B4. Gõ lệnh biên dịch: javac MyFirstJavaProgram.java

B5. Gõ lệnh thực thi: java MyFirstJavaProgram

```
E:\java>javac MyFirstJavaProgram.java  
E:\java>java MyFirstJavaProgram  
Hello World  
E:\java>
```



Chương trình JAVA



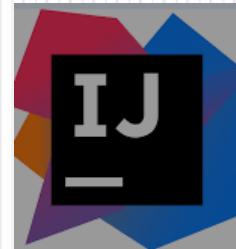


Chương trình JAVA

- ❑ **Viết mã nguồn:** dùng một chương trình soạn thảo văn bản, chẳng hạn như NotePad để viết mã nguồn và lưu lại thành tập tin có phần mở rộng “.java”.
- ❑ **Biên dịch ra mã bytecode:** dùng trình biên dịch Java (lệnh javac) để biên dịch mã nguồn “.java” thành mã bytecode java có phần mở rộng “.class”.
- ❑ **Thông dịch và thực thi:** Dùng trình thông dịch Java (lệnh java) để thông dịch và thực thi.

1.7. IDE hỗ trợ lập trình Java

- ❑ Một số IDE (*Integrated Development Environment*) hỗ trợ lập trình Java được sử dụng phổ biến:
 - ❑ Eclipse
 - ❑ NetBean
 - ❑ IntelliJ IDEA





2. Các thành phần cơ bản của NNLT Java

- 2.1. Từ khóa và định danh
- 2.2. Kiểu dữ liệu
- 2.3. Toán tử
- 2.4. Biến và hằng
- 2.5. Câu lệnh và khối lệnh
- 2.6. Chú thích
- 2.7. Nhập xuất dữ liệu đơn giản



2.1. Từ khóa và định danh

❑ **Từ khóa**

- ❑ Từ khóa của một ngôn ngữ lập trình là những định danh được định nghĩa trước của ngôn ngữ và không được phép sử dụng cho các thực thể khác.
- ❑ Các từ khóa của Java có thể chia thành **9** nhóm.



2.1. Từ khóa và định danh

❑ **Từ khóa**

- ❑ Tổ chức các lớp: package, import
- ❑ Định nghĩa các lớp: interface, class, extends, implements.
- ❑ Các từ khóa cho các biến và các lớp: abstract, public, private, protected, static, synchronized, volatile, final, native.
- ❑ Các kiểu dữ liệu cơ sở: long, int, short, byte, char, float, double, boolean, void.
- ❑ Các từ khóa cho các giá trị và các biến: false, true, this, super, null.



2.1. Từ khóa và định danh

❑ **Từ khóa**

- ❑ Xử lý ngoại lệ: throw, throws, try, catch, finally.
- ❑ Tạo lập và kiểm tra các đối tượng: new, instanceof.
- ❑ Dòng điều khiển chương trình: if, else, switch, case, default, break, continue, return, do, while, for.
- ❑ Một số từ khóa khác: byvalue, future, outer, const, genetic, rest, goto, inner, var, cast, operator.



2.1. Từ khóa và định danh

❑ Định danh (Identifier)

- ❑ Là tên gọi của các thành phần trong chương trình. Định danh được sử dụng để xác định các phần tử như biến, phương thức (method), đối tượng, lớp,...
- ❑ Trong Java, định danh là một dãy các ký tự gồm các chữ cái, chữ số, ký hiệu gạch nối “_”, các ký hiệu tiền tệ như \$, £,... không được bắt đầu bằng chữ số và không được trùng với từ khóa và các từ dành riêng của Java.



2.2. Kiểu dữ liệu

1. Kiểu dữ liệu cơ sở
2. Kiểu dữ liệu tham chiếu



2.2.1. Các kiểu dữ liệu cơ sở

Kiểu dữ liệu	Kích thước (bit)	Giá trị mặc định	Miền giá trị
byte	8	0	-128 đến 127
short	16	0	-32768 đến 32767
int	32	0	- 2^{31} đến $2^{31}-1$
long	64	0L	- 2^{63} đến $2^{63}-1$
float	32	0.0f	-3.40292347E+38 đến +3.40292347E+38
double	64	0.0d	-1,79769313486231570E+308 đến +1,79769313486231570E+308
boolean	1	false	true hoặc false
char	16	\u0000	'\u0000' to '\uffff'



2.2.2. Kiểu dữ liệu tham chiếu

- Array
- Class
- Interface



2.3. Toán tử

- Toán tử số học
- Toán tử nhị phân
- Toán tử quan hệ
- Toán tử luận lý (logic)
- Một số toán tử khác



2.3.1. Toán tử số học

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
-	Phép toán trừ	Số thực hoặc số nguyên	<code>int a, b; float x, y; 5 - 2.6; a - 3;</code>
+	Phép toán cộng	Số thực hoặc số nguyên	<code>6 + 5; a + y; 5.6 + 2.3; 5-1.2;</code>
*	Phép toán nhân	Số thực hoặc số nguyên	<code>a * x; b * y; 2.5 * 6.3;</code>



2.3.1. Toán tử số học (2)

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
/	Phép toán chia	Số thực hoặc số nguyên	$10.0 / 3.0 \rightarrow 3.33$ $10.0 / 3 \rightarrow 3.33$ $10 / 3.0 \rightarrow 3.33$
/	Phép chia lấy phần nguyên	Giữa 2 số nguyên	$10 / 3 \rightarrow 3$
%	Phép chia lấy phần dư	Giữa 2 số nguyên	$10 \% 3 \rightarrow 1$



2.3.2. Toán tử nhị phân

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
&	Phép và nhị phân	2 số nhị phân	$0 \& 0 \rightarrow 0$ $0 \& 1 \rightarrow 0$ $1 \& 0 \rightarrow 0$ $1 \& 1 \rightarrow 1$
	Phép hoặc nhị phân	2 số nhị phân	$0 0 \rightarrow 0$ $0 1 \rightarrow 1$ $1 0 \rightarrow 1$ $1 1 \rightarrow 1$



2.3.2. Toán tử nhị phân (2)

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
\wedge	Phép HOẶC có loại trừ	2 số nhị phân	$0 \mid 0 \rightarrow 0$ $0 \mid 1 \rightarrow 1$ $1 \mid 0 \rightarrow 1$ $1 \mid 1 \rightarrow 0$
$<<$	Phép dịch trái nhị phân	Số nhị phân	$a << n \rightarrow a * 2^n$ $101 << 2 \rightarrow 10100$
$>>$	Phép dịch phải nhị phân	Số nhị phân	$a >> n \rightarrow a / 2^n$ $101 >> 2 \rightarrow 1$
\sim	Phép đảo bit nhị phân	Số nhị phân	$\sim 0 \rightarrow 1$ $\sim 1 \rightarrow 0$



2.2.3. Toán tử nhị phân (3)

❑ Ví dụ:

```
int a = 27;           // 0001 1011
int b = 18;           // 0001 0010
int c1, c2, c3, c4, c5, c6;
c1 = a & b;          // 0001 0010
c2 = a | b;          // 0001 1011
c3 = a ^ b;          // 0000 1001
c4 = ~a;             // 1110 0100
c5 = a << 3;         // 1101 1000
c6 = a >> 3;         // 0000 0011
```



2.2.4. Toán tử quan hệ

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
>	So sánh lớn hơn	2 số nguyên hoặc số thực	$1 > 2 \rightarrow 0$ $5 > 4 \rightarrow 1$
\geq	So sánh lớn hơn hoặc bằng	2 số nguyên hoặc số thực	$2 \geq 2 \rightarrow 1$ $3 \geq 4 \rightarrow 0$
<	So sánh nhỏ hơn	2 số nguyên hoặc số thực	$1 < 2 \rightarrow 1$ $5 < 4 \rightarrow 0$
\leq	So sánh nhỏ hơn hoặc bằng	2 số nguyên hoặc số thực	$2 \leq 2 \rightarrow 1$ $4 \leq 3 \rightarrow 0$
\equiv	So sánh bằng nhau	2 số nguyên hoặc số thực	$4 \equiv 5 \rightarrow 0$
\neq	So sánh không bằng nhau (khác)	2 số nguyên hoặc số thực	$2 \neq 2 \rightarrow 0$ $1 \neq 2 \rightarrow 1$



2.2.5. Toán tử logic

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ
$\&\&$	Phép và logic,	Hai biểu thức logic	$2 < 3 \&\& 4 < 5 \rightarrow 1$ $2 < 1 \&\& 4 < 5 \rightarrow 0$
\parallel	Phép hoặc logic	Hai biểu thức logic	$2 < 3 \parallel 4 < 5 \rightarrow 1$ $2 < 1 \parallel 5 < 4 \rightarrow 0$
!	Phép phủ định logic	Biểu thức logic	$!1 \rightarrow 0$ $!(2 > 3) \rightarrow 1$



2.2.6. Toán tử điều kiện

- ❑ Toán tử điều kiện: ? :
<Biểu thức logic>?<Giá trị 1>:<Giá trị 2>

- ❑ Nếu biểu thức logic đúng thì kết quả
<là giá trị 1> ngược lại kết quả là giá
 int x, y;
 y = 4;
 ❑ x = (y > 5) ? 25 : 50; int x, y;
 y = 7;
 x = (y > 5) ? 25 : 50;



2.3. Biến và hằng

❑ Biến

❑ Cú pháp khai báo biến:

<Kiểu dữ liệu> <tên biến>;

❑ Ví dụ:

int a, b;

❑ Hằng

❑ Cú pháp khai báo hằng:

final <Kiểu dữ liệu> <tên hằng> = <giá trị>;

❑ Ví dụ:

final double PI = 3.1416;



2.4. Câu lệnh và khối lệnh

- ❑ Các câu lệnh trong java được kết thúc bằng dấu chấm phẩy (;
- ❑ Một khối lệnh là một nhóm từ hai lệnh trở lên, được bao bằng cặp dấu {} và được thực hiện tuần tự. Bên trong một khối lệnh có thể chứa một hay nhiều lệnh hoặc khối lệnh khác.



2.4. Câu lệnh và khối lệnh

```
{ //bắt đầu khối lệnh 1
    //lệnh 1.1
    { //bắt đầu khối lệnh 2
        //lệnh 2.1
        //lệnh 2.2
        ...
    } //kết thúc khối lệnh 2
    //lệnh 2.2
    ...
} //kết thúc khối lệnh 1
{//bắt đầu khối lệnh 3
    //các lệnh thuộc khối lệnh 3
}
```



2.5. Chú thích

- ❑ Java cung cấp 3 loại chú thích:
 - ❑ Chú thích trên một dòng: ghi nội dung chú thích sau dấu //
//Từ đây đến cuối dòng là
chú thích
 - ❑ Chú thích trên nhiều dòng: ghi nội dung chú thích giữa dấu /* và */
/* Chú thích được ghi
trên nhiều dòng
*/



2.5. Chú thích (4)

❑ Chú thích trong tư liệu (Javadoc)

/* *

* Phương thức này dùng để
cộng 2 số

* Tác giả: NV A

* Version 0.1

* /



2.5. Chú thích (3)

- ❑ Chú thích trong tư liệu (javadoc): là loại chú thích đặt biệt được đặt vào những chỗ thích hợp trong chương trình để javadoc có thể đọc và sử dụng tạo ra tư liệu dạng HTML cho chương trình.
- ❑ Thường đặt vào trước phần định nghĩa các lớp, interface, phương thức và biến. Phần chú thích trong tư liệu được bắt đầu bằng `/**` và kết thúc bằng `*/`.

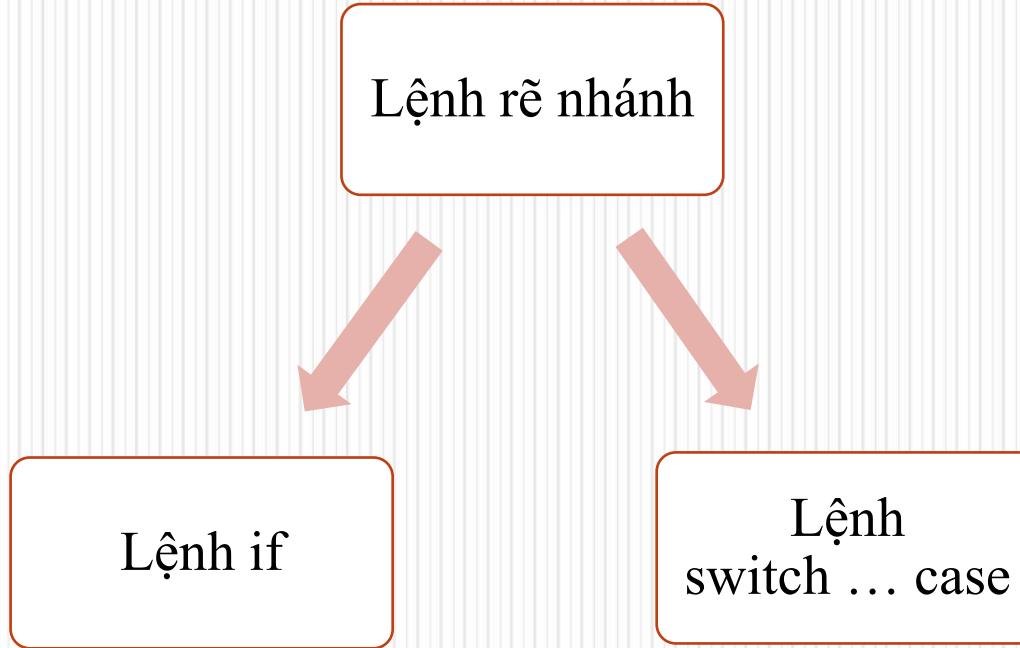


3. Lệnh điều khiển

1. Lệnh rẽ nhánh
2. Lệnh lặp

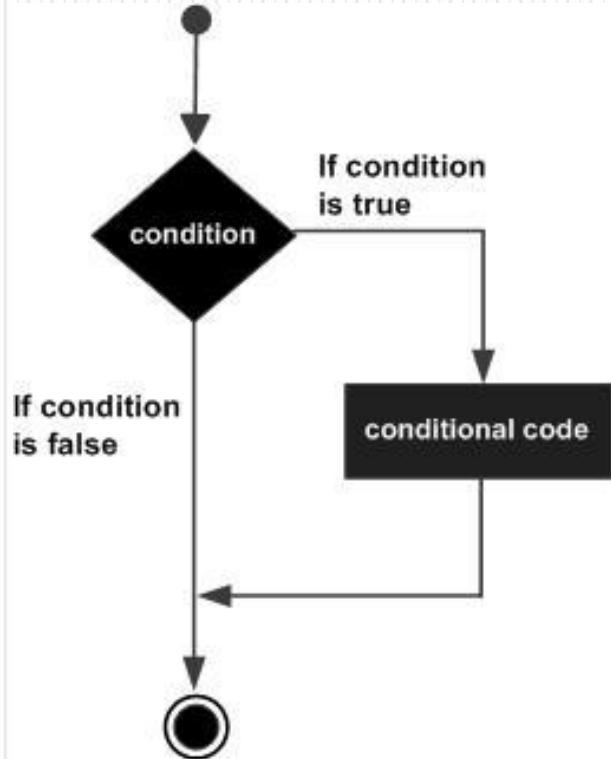


3.1. Lệnh rẽ nhánh



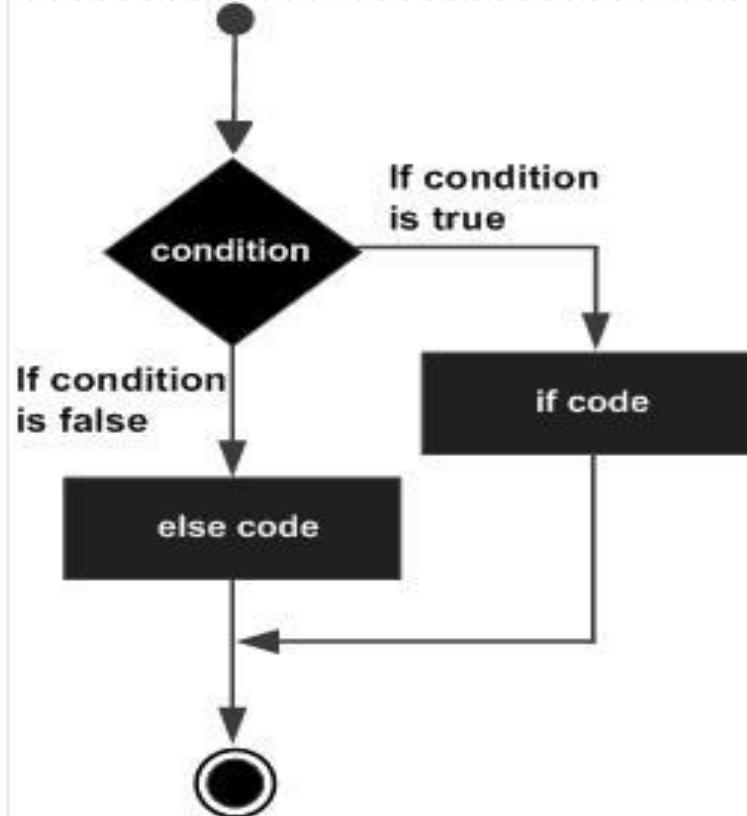


3.1.1. Lệnh if



boolean expression
↓
`if (x > y)`
{
sequence of statements → int t = x;
x = y;
y = t;
}

3.1.2. Lệnh if ... else





Lệnh if, if...else

<i>absolute value</i>	<pre>if (x < 0) x = -x;</pre>
<i>put the smaller value in x and the larger value in y</i>	<pre>if (x > y) { int t = x; x = y; y = t; }</pre>
<i>maximum of x and y</i>	<pre>if (x > y) max = x; else max = y;</pre>
<i>error check for division operation</i>	<pre>if (den == 0) System.out.println("Division by zero"); else System.out.println("Quotient = " + num/den);</pre>
<i>error check for quadratic formula</i>	<pre>double discriminant = b*b - 4.0*c; if (discriminant < 0.0) { System.out.println("No real roots"); } else { System.out.println((-b + Math.sqrt(discriminant))/2.0); System.out.println((-b - Math.sqrt(discriminant))/2.0); }</pre>



3.1.3. Lệnh if ... else lồng nhau

❑ Cú pháp

```
if (<điều kiện 1>){  
    <lệnh/khối lệnh 1>;  
}  
else if (<điều kiện 2>){  
    <lệnh/khối lệnh 2>;  
}  
...  
else if (<điều kiện n>){  
    <lệnh/khối lệnh n>;  
}  
else{  
    <lệnh/khối lệnh khác>;  
}
```



3.1.3. Lệnh if ... else lồng nhau

□ Ví dụ

```
if (diem>=9){  
    System.out.println("Xuat sac");  
} else if (diem>=8){  
    System.out.println("Gioi");  
} else if (diem>=7){  
    System.out.println("Kha");  
} else if (diem>=5){  
    System.out.println("Trung binh");  
}  
else{  
    System.out.println("Duoi trung binh");  
}
```



3.1.3. Lệnh if ... else lồng nhau

```
public static void main(String[] args) {
    int a=4,b=5,c=9;
    if(a!=0)
    {
        float delta= b*b-4*a*c;
        if(delta>0){
            System.out.print("Phương trình có 2 nghiệm phân biệt:" + ((-b+
                Math.sqrt(delta))/(2*a)) + ";" + ((-b- Math.sqrt(delta))/(2*a)));
        }
        else
            if(delta==0){
                System.out.print(" Phương trình có 1 nghiệm kép:" + (-b/(2*a)));
            }
            else System.out.print(" Phương trình vô nghiệm");
    }
    else
        if(b==0)
            if(c==0) System.out.print(" Phương trình vô số nghiệm");
            else      System.out.print(" Phương trình vô nghiệm");
        else System.out.print(" Phương trình có 1 nghiệm là: " + -c/b);
    }
}
```



3.1.4. Lệnh switch ... case

❑ Cú pháp

```
switch (<biểu thức>) {  
    case <giá trị 1>:  
        <lệnh/khối lệnh 1>;  
        break;  
    case <giá trị 2>:  
        <lệnh/khối lệnh 2>;  
        break;  
    ...  
    case <giá trị n>:  
        <lệnh/khối lệnh n>;  
        break;  
    default:  
        <lệnh/khối lệnh  
        default>;  
}
```



3.1.4. Lệnh switch ... case

```
public static void main (String[] args) {  
    int thang = 11;  
    switch(thang) {  
        case 1: System.out.println("Tháng 1"); break;  
        case 2: System.out.println("Tháng 2"); break;  
        case 3: System.out.println("Tháng 3"); break;  
        case 4: System.out.println("Tháng 4"); break;  
        case 5: System.out.println("Tháng 5"); break;  
        case 6: System.out.println("Tháng 6"); break;  
        case 7: System.out.println("Tháng 7"); break;  
        case 8: System.out.println("Tháng 8"); break;  
        case 9: System.out.println("Tháng 9"); break;  
        case 10: System.out.println("Tháng 10"); break;  
        case 11: System.out.println("Tháng 11"); break;  
        case 12: System.out.println("Tháng 12"); break;  
        default: System.out.println("Tháng không hợp lệ"); break;  
    }  
}
```



3.1.4. Lệnh switch ... case

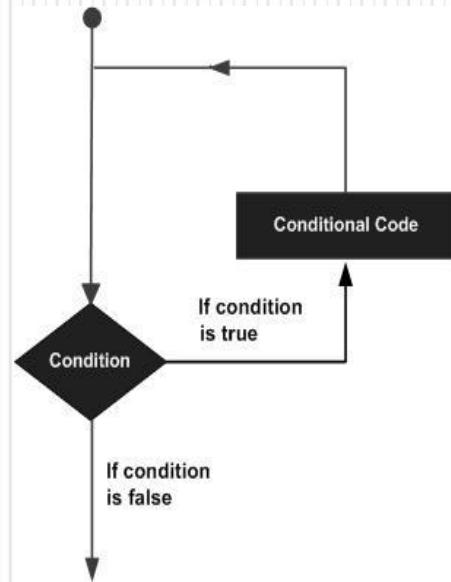
The screenshot shows a Java code editor with the file 'Test20.java' open. The code contains a switch statement that prints different messages based on a grade value. A vertical red line is drawn on the left margin of the code editor, highlighting the entire switch block from its opening brace to its closing brace. The code is as follows:

```
1 public class Test20 {
2
3     public static void main(String args[]) {
4         // char grade = args[0].charAt(0);
5         char grade = 'C';
6
7         switch(grade) {
8             case 'A' :
9                 System.out.println("Excellent!");
10            break;
11            case 'B' :
12            case 'C' :
13                System.out.println("Well done");
14                break;
15            case 'D' :
16                System.out.println("You passed");
17            case 'F' :
18                System.out.println("Better try again");
19                break;
20            default :
21                System.out.println("Invalid grade");
22        }
23        System.out.println("Your grade is " + grade);
24    }
25 }
```



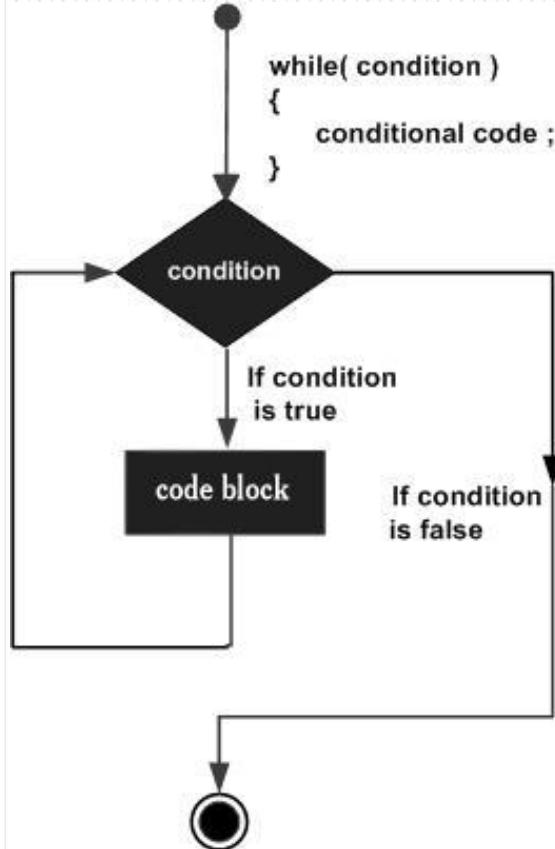
3.2. Lệnh lặp

- 1. Lệnh while**
- 2. Lệnh do ... while**
- 3. Lệnh for**
- 4. Lệnh break**
- 5. Lệnh continue**





3.2.1. Lệnh while



while (<biểu thức điều kiện>) {

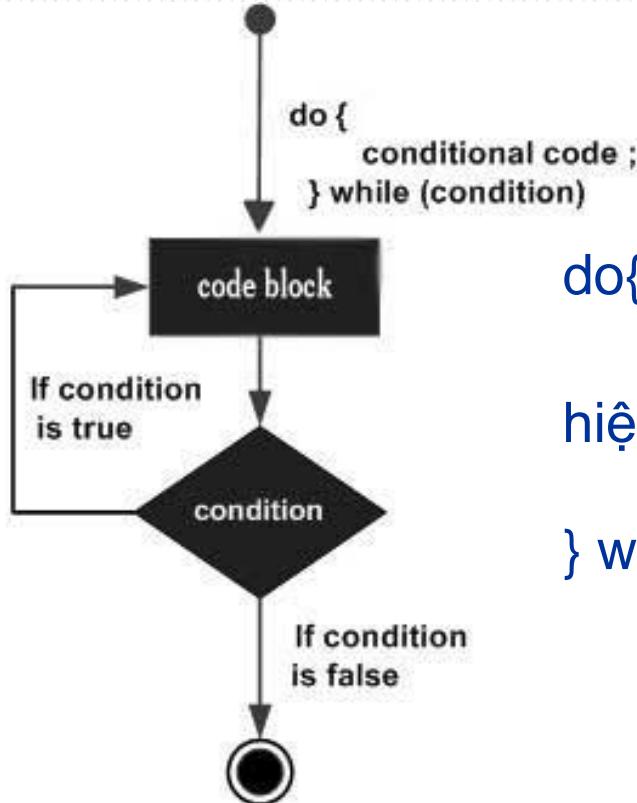
[Các câu lệnh thực

hiện];

}
initialization is a
separate statement
loop-
continuation
condition
int power = 1;
while (power <= n/2)
{
 power = 2*power;
}
braces are
optional
when body
is a single
statement
body



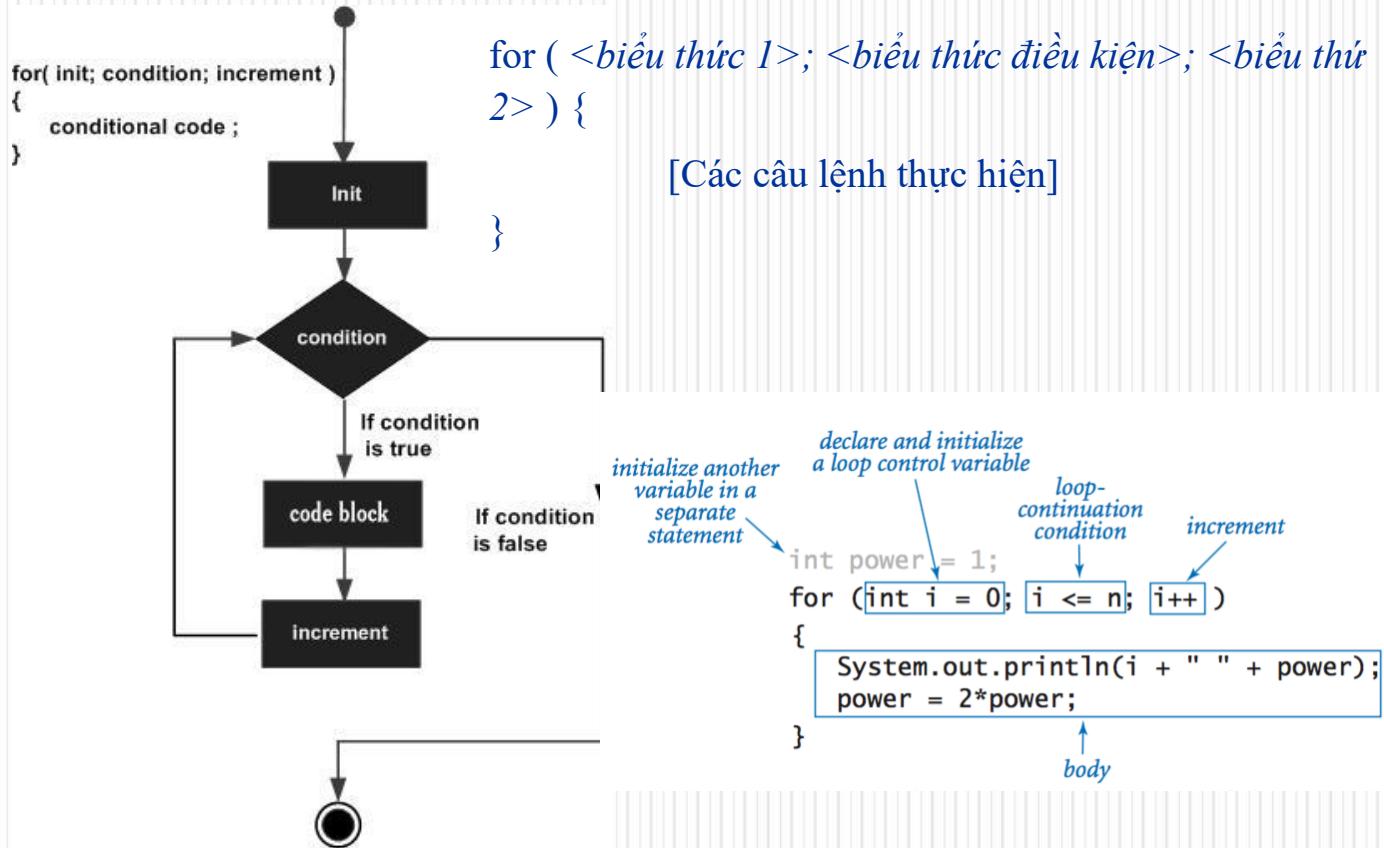
3.2.2. Lệnh do ... while



do{
 [Các câu lệnh thực
hiện];
} while (<biểu thức điều kiện>);

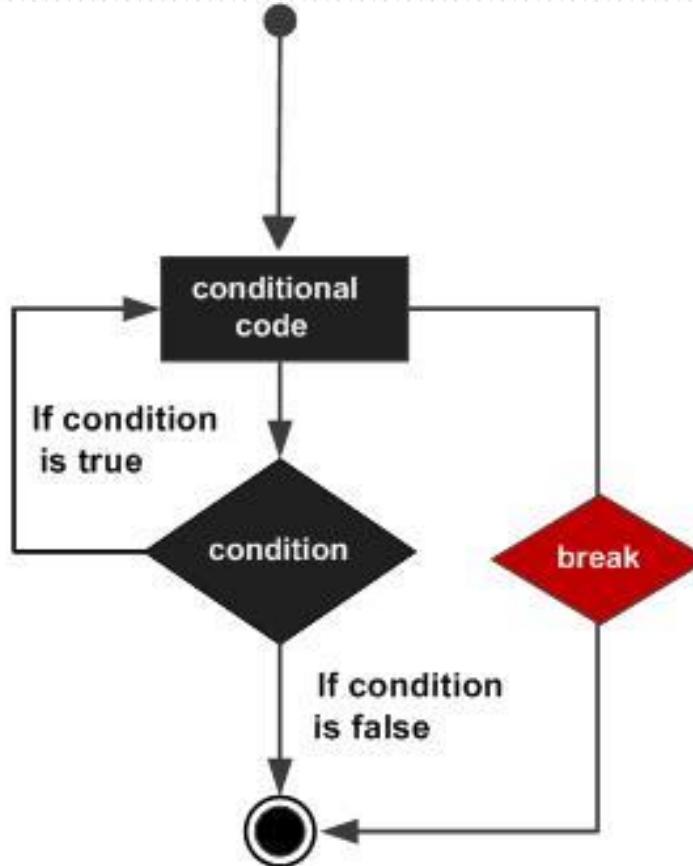


3.2.3. Lệnh for





3.2.4. Lệnh break





3.2.4. Lệnh break

The image shows a Java code editor and a terminal window. The code editor displays the file `Test13.java` with the following content:

```
1 public class Test13 {
2
3     public static void main(String args[]) {
4         int [] numbers = {10, 20, 30, 40, 50};
5
6         for(int x : numbers ) {
7             if( x == 30 ) {
8                 break;
9             }
10            System.out.print( x );
11            System.out.print("\n");
12        }
13    }
14 }
```

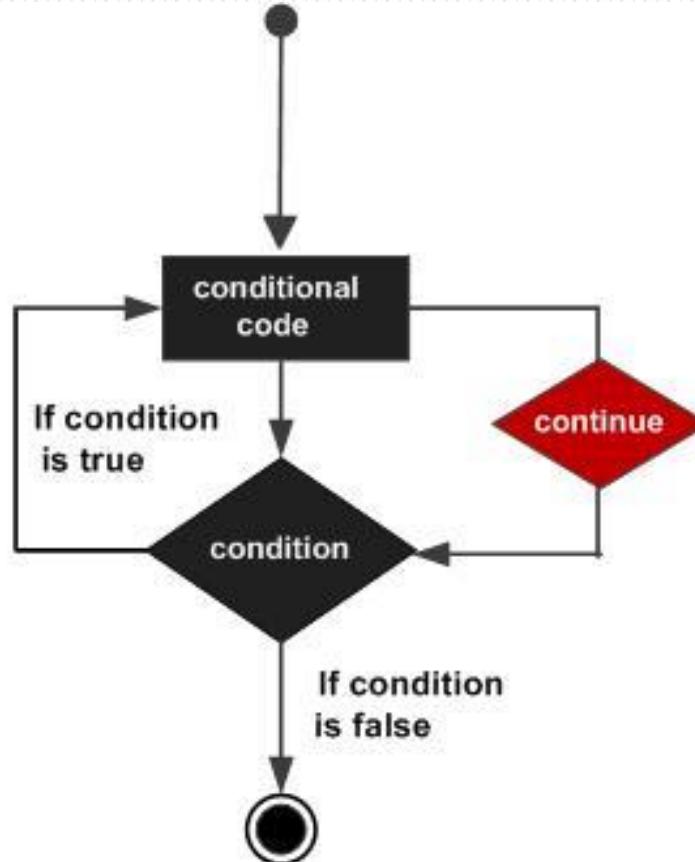
The terminal window below shows the command-line interface with the following session:

```
C:\WINDOWS\system32\cmd.exe
E:\java>javac Test13.java
E:\java>java Test13
10
20

E:\java>
```



3.2.5. Lệnh continue





3.2.5. Lệnh continue

The screenshot shows a Java code editor window titled "Test14.java" and a terminal window below it.

Java Code (Test14.java):

```
1  public class Test14 {
2
3      public static void main(String args[]) {
4          int [] numbers = {10, 20, 30, 40, 50};
5
6          for(int x : numbers ) {
7              if( x == 30 ) {
8                  continue;
9              }
10             System.out.print( x );
11             System.out.print("\n");
12         }
13     }
14 }
```

Terminal Output:

```
C:\WINDOWS\system32\cmd.exe
E:\java>javac Test14.java
E:\java>java Test14
10
20
40
50
E:\java>
```



4. Mảng và chuỗi

4.1. Chuỗi ký tự - lớp String và String Builder

4.2. Mảng một chiều

4.3. Mảng hai chiều



4.1. Chuỗi ký tự - Lớp String

□ Khai báo

□ Cách 1: **String** <tên biến> = “Chuỗi ký tự”;

□ Cách 2:

String <tên biến> = new **String**(“Chuỗi ký tự”);

□ Ví dụ:

□ **String** s1 = “Hello”;

□ **String** s2 = new **String**(“Hello”);



4.1. Chuỗi ký tự - Lớp String (2)

Một số phương thức thông dụng

❑ int **length()**

- ❑ Trả về số ký tự của chuỗi
- ❑ Ví dụ: **String** s = “Hello Java”
- ❑ **int** n = s.length(); → n = 10

❑ boolean **equals(String str)**

- ❑ Trả về true nếu 2 chuỗi giống hệt nhau,
false nếu 2 chuỗi khác nhau.

❑ **boolean kt = s.equals(“Hello Java”);** → kt=true



4.1. Chuỗi ký tự - Lớp String (3)

Một số phương thức thông dụng

❑ **String trim()**

❑ Trả về chuỗi sau khi cắt các khoảng trắng thừa.

❑ **boolean isEmpty()**

❑ Trả về true nếu chuỗi là một chuỗi rỗng ($length = 0$), ngược lại là false.

❑ **boolean kt = s.isEmpty(); → kt = fasle**



4.1. Chuỗi ký tự - Lớp String (4)

Một số phương thức thông dụng

❑ **String toUpperCase()**

- ❑ Trả về chuỗi sau khi chuyển toàn bộ sang chữ hoa.
 - ❑ s.toUpperCase(); → s = “HELLO JAVA”

❑ **String toLowerCase()**

- ❑ Trả về chuỗi sau khi chuyển toàn bộ sang chữ thường.
 - ❑ s.toLowerCase(); → s = “hello java”



4.1. Chuỗi ký tự - Lớp String (5)

Một số phương thức thông dụng

❑ **char charAt(int i)**

- ❑ Trả về ký tự thứ i trong chuỗi.
- ❑ Chars ch = s.charAt(1); → ch = “e”

❑ **int compareTo(String str)**

- ❑ So sánh từng ký tự của 2 chuỗi bằng mã ASCII.
Trả về 0 nếu 2 chuỗi giống hệt nhau; <0 nếu chuỗi 1 < chuỗi 2; >0 nếu chuỗi 1 > chuỗi 2.
- ❑ int kt = s.compareTo("Gello"); → kt = 1



4.11. Chuỗi ký tự - Lớp String (5)

- ❑ Lớp **StringBuilder**: sinh viên tự tìm hiểu.
- ❑ **Bài tập. Viết chương trình nhập vào một chuỗi và in ra:**
 - ❑ Các ký tự viết hoa có trong chuỗi.
 - ❑ Tổng giá trị các ký tự là số có trong chuỗi.
 - ❑ Chuỗi sau khi chuyển toàn bộ thành chữ hoa.



4.2. Mảng một chiều

❑ Khái niệm:

- ❑ Mảng là một dãy các phần tử có cùng kiểu dữ liệu và cùng tên.
- ❑ Các phần tử được phân biệt qua chỉ số.

❑ Khai báo

❑ Cách 1: **KieuDuLieu[] tenBien;**

❑ Cách 2: **KieuDuLieu ten_a**



Lưu ý: Chỉ số mảng bắt đầu từ **0**.



4.2. Mảng một chiều

❑ Khai báo

- ❑ `int[] arr1; //Khai báo mảng kiểu int`
- ❑ `long[] arr2; //Khai báo mảng kiểu long`
- ❑ `float[] arr3; //Khai báo mảng kiểu float`
- ❑ `double[] arr4; //Khai báo mảng kiểu double`
- ❑ `String[] arr5 ; //Khai báo mảng kiểu string`
- ❑ `PhanSo[] arr6 ; //Khai báo mảng kiểu PhanSo`
- ❑ `SinhVien[] arr7 ; //Khai báo mảng kiểu PhanSo`



4.2. Mảng một chiều

❑ Khai báo và cấp phát vùng nhớ

KieuDuLieu[] tenBien = new KieuDuLieu [n];

hoặc

KieuDuLieu[] tenBien; tenBien = new KieuDuLieu [n];

❑ Ví dụ:

- ❑ float[] arr3 = new float [7]; String[] arr6 = new String [6];
- ❑ SinhVien[] dsSV = new SinhVien[100];
- ❑ PhanSo[] dsPS; dsPS = new PhanSo[10];



4.2. Mảng một chiều

❑ Khởi tạo mảng

❑ Cách 1

```
int[] arr = {1, 3, 5, 7, 9}; //arr.length = 5
```

❑ Cách 2

```
int [] arr = new int[5];
```

```
arr[0]=1;
```

```
arr[1]=3;
```

```
arr[2]=5;
```

```
arr[3]=7;
```

```
arr[4]=9;
```



4.2. Mảng một chiều

□ Khởi tạo mảng

long[] arr1 = {1, 3, 5, 7, 9};

float[] arr2 = { (float) 1.3, (float) 3.2, (float) 5.5};

double[] arr2 = {2.3, 7.2, 9.5}

String[] ngay = {

"chủ nhật", "thứ hai", "thứ ba",

"thứ tư", "thứ năm", "thứ sáu", "thứ bảy" };



4.2. Mảng một chiều

❑ Xuất mảng

- ❑ Cách 1: sử dụng cấu trúc lặp **foreach**

//tenBien lần lượt là các phần tử bên trong mảng

```
for (KieuDuLieu tenBien : tenMang){
```

Các câu lệnh;

```
}
```

- ❑ Ví dụ: Xuất các phần tử của mảng các số nguyên a

```
System.out.println("Xuất mảng dùng foreach");
```

```
System.out.println("Số phần tử của mảng " + a.length);
```

```
for (int pt : a){
```

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

```
}
```



4.2. Mảng một chiều

❑ Xuất mảng

❑ Cách 2: Sử dụng cấu trúc lặp *for* để xuất mảng

❑ Ví dụ:

```
System.out.println("Xuất mảng dùng for ");
```

```
System.out.println("Số phần tử của mảng " + a.length);
```

```
for (int i=0 ; i<a.length; i++) {
```

```
    System.out.println(a[i]);
```

```
}
```



4.2. Mảng một chiều

□ Nhập mảng

System.out.print("Số phần tử của mảng là ");

Scanner input = new Scanner(System.in);

int n = input.nextInt();

int [] a = new int [n]; //a.length = n

for (int i = 0; i < a.length; i++) {

 System.out.print("a["+i+"]=");

 a[i] = input.nextInt();

}



4.2. Mảng một chiều

❑ Xuất mảng mảng

System.out.println("Số phần tử của mảng " + a.length);

//Xuất mảng dùng for

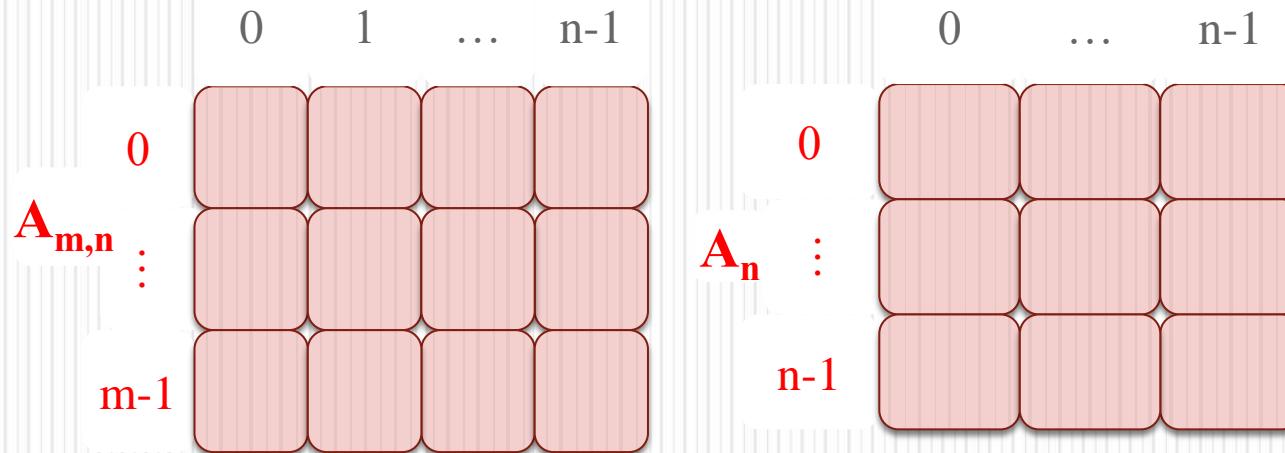
```
for (int i = 0; i < a.length; i++) {  
    System.out.println(a[i]);
```

}

//Xuất mảng dùng foreach

```
for (int pt : a) {  
    System.out.println(pt);  
}
```

4.3. Mảng hai chiều





4.3. Mảng hai chiều

❑ Khai báo

- ❑ KieuDuLieu [][] tenBien;
- ❑ Hoặc KieuDuLieu tenBien [][];

❑ Ví dụ

```
//Khai báo mảng hai chiều kiểu int  
int[][] arr1;  
//Khai báo mảng hai chiều kiểu long  
long[][] arr2;  
//Khai báo mảng hai chiều kiểu float  
float[][] arr3;
```



4.3. Mảng hai chiều

❑ Khai báo và cấp phát vùng nhớ

❑ Cách 1:

KieuDuLieu[][] tenMang = new KieuDuLieu [m][n];

❑ Cách 2:

KieuDuLieu[][] tenMang;

tenMang = new KieuDuLieu [m][n];

❑ m: số dòng (tenMang.length)

❑ n: số cột (tenMang[i].length)



4.3. Mảng hai chiều

❑ Khai báo và cấp phát vùng nhớ

//Khai báo và cấp phát mảng hai chiều kiểu int

```
int[][] arr1 = new int[3][5];
```

```
int soDong = a.length; //soDong = 3
```

```
int soCot = a[i].length; //soCot = 5
```

//Khai báo và cấp phát mảng kiểu long

```
long[][] arr2 = new long[5][6];
```

```
int soDong = a.length; //soDong = 5
```

```
int soCot = a[i].length;; //soCot = 6
```



4.3. Mảng hai chiều

❑ Khởi tạo giá trị

//Cách 1

```
int[][] a = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

```
int soDong = a.length; //soDong = 4
```

```
int soCot = a[i].length; //soCot = 2
```



4.3. Mảng hai chiều

□ Khởi tạo giá trị

//Cách 2

```
int [][] a = new int[4][2];
int soDong = a.length; //soDong = 4
int soCot = a[0].length; //soCot = 2
int k = 1;
for (int i = 0; i < soDong; i++) {
    for (int j = 0; j < soCot; j++) {
        a[i][j]=k++;
    }
}
```



4.3. Mảng hai chiều

❑ Nhập mảng 2 chiều

//Nhập số dòng: m

//Nhập số cột: n

```
int[][] a=new int[m][n];
```

```
for (int i = 0; i < a.length; i++){
```

```
    for (int j = 0; j < a[i].length; j++) {
```

```
        System.out.print("a["+i+"]["+j+"]=");
```

```
        a[i][j] = input.nextInt();
```

```
}
```

```
}
```



4.3. Mảng hai chiều

❑ Xuất mảng 2 chiều

```
System.out.println("Xuất mảng");
System.out.println("Số dòng :" + a.length);
System.out.println("Số cột : " + a[0].length);
for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < a[i].length; j++) {
        System.out.print(a[i][j]+"\t");
    }
    System.out.println();
}
```



5. Nhập xuất dữ liệu đơn giản

5.1. Nhập dữ liệu từ bàn phím

5.2 Xuất dữ liệu ra màn hình



5.1. Nhập dữ liệu từ bàn phím

- ❑ Import gói java.util.Scanner vào đầu chương trình

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

- ❑ Tạo đối tượng thuộc lớp Scanner

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

- ❑ Sử dụng đối tượng scanner gọi phương thức đọc dữ liệu phù hợp với yêu cầu.

- ❑ Ví dụ: Đọc dòng văn bản

```
String str =  
input.nextLine();
```



5.1. Nhập dữ liệu từ bàn phím

- ❑ Tương tự, để đọc các giá trị số nguyên, số thực có thể dùng:

```
int n = input.nextInt();
```

```
float f = input.nextFloat();
```



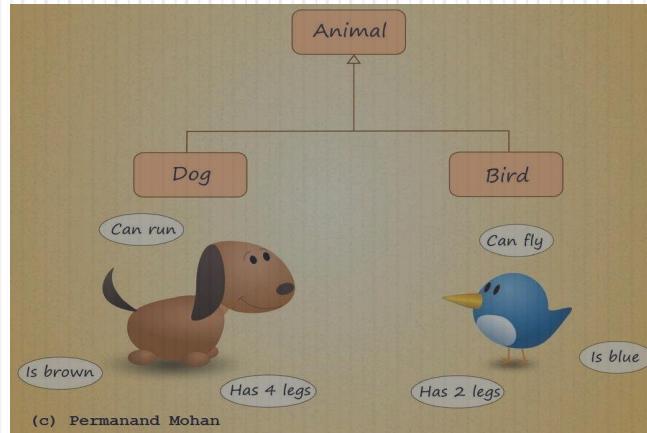
5.2. Xuất dữ liệu ra màn hình

- ❑ Sử dụng System.out.print(""); hoặc System.out.println("");
- ❑ Ví dụ:
 - ❑ System.out.print ("Hello World");
 - ❑ String s = "Hello World";
 - ❑ System.out.println(s);

6. Lập trình hướng đối tượng trong Java

Object-oriented programming is a **method** of implementation in which programs are organized as **cooperative collections of objects**, each of which represents an **instance of some class**, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

(Grady Booch et al)





Lập trình cỗ điện và Lập trình HĐT

Lập trình cỗ điện

Chương trình

```
typedef struct {
    char *mssv;
    char *hoten;
    float diemTB;
    ...
} Sinhvien;

Sinhvien *dssv1, *dssv2;

void sapxep(Sinhvien *) {...}
void luu(Sinhvien *, char *) {...}
void nhap(Sinhvien *) {...}

void main() { nhap(dssv1);
              sapxep(dssv2);...}
```

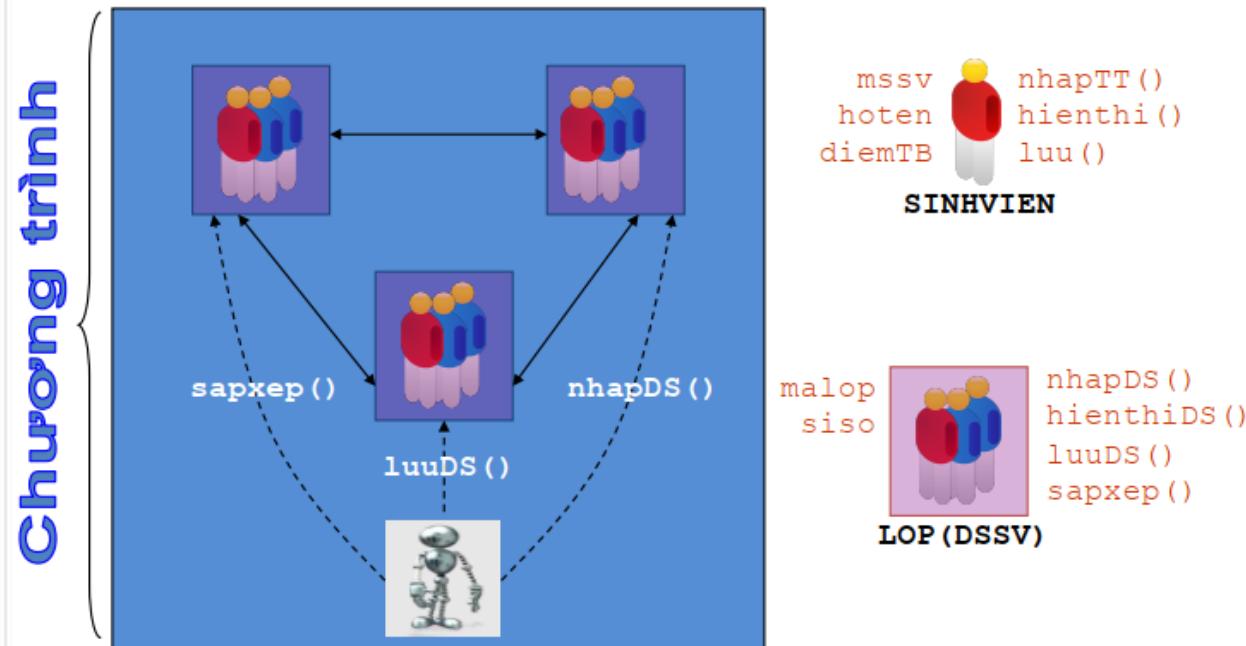
Cấu trúc dữ liệu

Giải thuật



Lập trình cổ điển và Lập trình HĐT

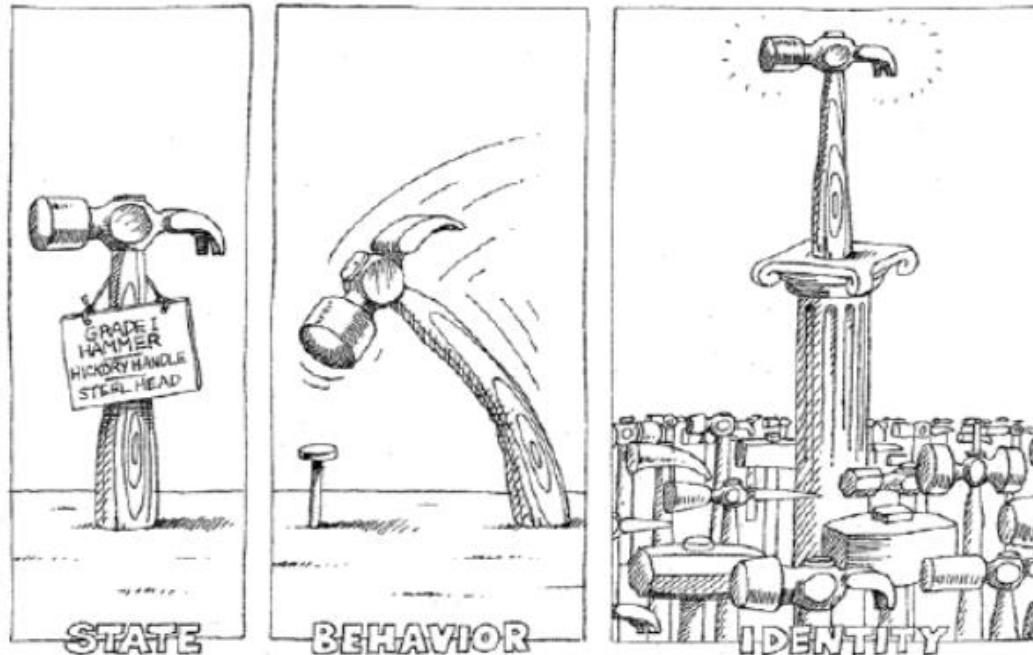
Lập trình hướng đối tượng





6.1. Đối tượng và lớp

❑Đối tượng (Object)



An object has state, exhibits some well-defined behavior,
and has a unique identity.

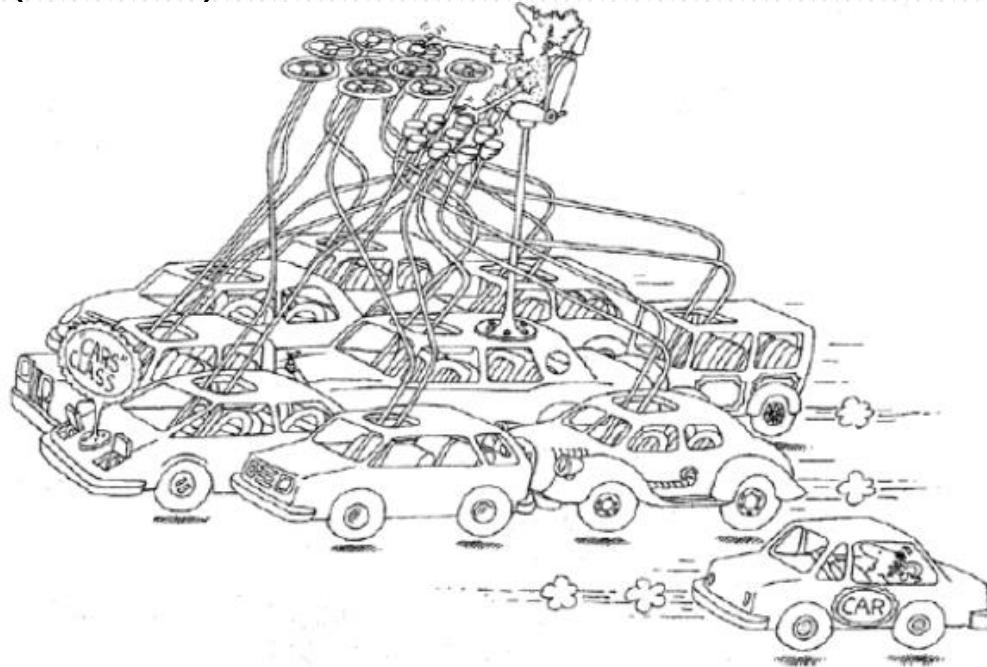
6.1. Đối tượng và lớp

☐ Đối tượng (Object)

Sự vật	Thuộc tính	Hành vi	Ví dụ
Con chó	Tên: Mino Màu sắc: Xám Giống: Nhật Trạng thái: Vui vẻ	Sửa Ăn Chạy Cắn	
Stack A	List Empty: NO.	PUSH POP	
Bóng đèn	Nhãn hiệu: ABC Màu: Xanh Trạng thái: Mở Loại: Đèn bàn	Bật Tắt Sáng Mờ	

6.1. Đối tượng và lớp

☐ Lớp (Class)

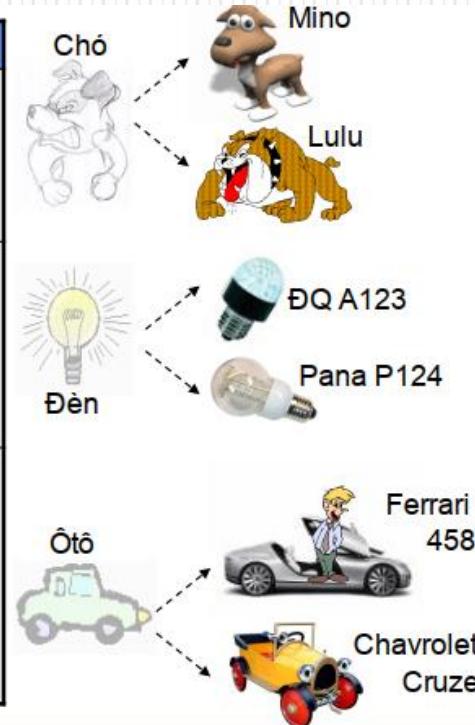


A class represents a set of objects that share a common structure and a common behavior.

6.1. Đối tượng và lớp

☐ Lớp (Class)

Lớp	Thuộc tính	Hành vi
Chó	Tên Màu sắc Giống Trạng thái	Sửa Ăn Chạy Cắn
Bóng đèn	Nhãn hiệu Màu Trạng thái Loại	Bật Tắt Sáng Mờ
Xe ôtô	Nhãn hiệu Màu sắc Giá Hộp số Tốc độ ...	Chạy Dừng Tăng tốc Giảm tốc ...





6.1.1. Xây dựng lớp



[<tùy khóa truy cập>] **class**
<TÊN LỚP>
{
 <thành phần thuộc tính>;
 <thành phần phương
thức>;
};

A yellow rectangular callout box is positioned above the closing brace "};". It contains the text "private", "protected", and "public" stacked vertically, with a horizontal line connecting each word to the brace.



6.1.1. Xây dựng lớp

□ Tạo phương thức

```
[<Tùy khóa truy cập>] <kiểu trả về> <tên  
phương thức>(<ds tham số>) {  
    // thân phương thức  
}
```

Ví dụ:

```
public float calculateCharge(float hours) {  
    return(hours * billingRate);  
}
```



6.1.1. Xây dựng lớp

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From any class outside the package	Yes	No	No	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes	No	No



6.1.1. Xây dựng lớp

❑ Ví dụ: tạo lớp Xây dựng lớp HìnhTron có thuộc tính bán kính và phương thức tính chu vi, tính diện tích.

```
3 public class HinhTron {  
4     private double bankinh;  
5     public double tinhChuVi() {  
6         return 2*Math.PI*this.bankinh;  
7     }  
8     public double tinhDienTich() {  
9         return Math.PI*Math.pow(this.bankinh,2);  
10    }  
11    public void xuat() {  
12        System.out.println("Bán kính hình tròn là: " + this.bankinh);  
13        System.out.println("Chu vi hình tròn là: " +this.tinhChuVi());  
14        System.out.println("Diện tích hình tròn là: " + this.tinhDienTich());  
15    }  
16 }
```



6.1.2. Khai báo và sử dụng đối tượng

- ❑ Khai báo đối tượng của lớp

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

Ví dụ: `HinhTron ht=new HinhTron();`

- ❑ Truy xuất thành phần của đối tượng

`<tên đối tượng>.<tên thành phần>;`

Ví dụ: Gọi phương thức chu vi của đối tượng
`ht.tinhChuVi();`



6.1.3. Phương thức get, set

□ Get

```
public <Kiểu dữ liệu> getX()
{
    return ...;
}
```

□ Set

```
public void setX(<Kiểu dữ liệu> <biến>)
{
    //gán giá trị cho field
}
```



6.1.3. Phương thức get, set

```
public class Diem2D {  
    private int x;  
    private int y;  
    public void setX(int x) {  
        this.x=x;  
    }  
    public int getX() {  
        return x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
}
```



6.1.4. Phương thức khởi tạo

- ❑ Phương thức khởi tạo dùng để khởi tạo giá trị ban đầu cho các thành phần dữ liệu của đối tượng.
- ❑ Đặc điểm của phương thức khởi tạo
 - ❑ Không có kiểu trả về;
 - ❑ Trùng tên với tên lớp;
 - ❑ Phạm vi truy xuất thường là public;
 - ❑ Một lớp có thể có nhiều phương thức khởi tạo;
 - ❑ Được tự động gọi khi đối tượng được tạo ra.



6.1.4. Phương thức khởi tạo

- ❑ Phương thức khởi tạo mặc định
- ❑ Phương thức khởi tạo có tham số
- ❑ Phương thức khởi tạo sao chép



6.1.4. Phương thức khởi tạo

```
public class Diem2D {  
    int x;  
    int y;  
    public Diem2D()  
    {  
        x=0;  
        y=0;  
    }  
    public Diem2D(int x, int y)  
    {  
        this.x=x;  
        this.y=y;  
    }  
}
```

Phương thức khởi
tạo không tham
số
(PTKT mặc định)

```
public static void main(String[] args) {  
    //tạo điểm có tọa độ (0,0)  
    Diem2D diemA=new Diem2D();  
    //tạo điểm có tọa độ (3,4)  
    Diem2D diemB=new Diem2D(3, 4);  
}
```

Phương thức khởi
tạo có tham số
(2 tham số)



6.1.4. Phương thức khởi tạo

```
class Clock {  
    private h, m, s;  
  
    public Clock() {  
        h = m = s = 0;  
    }  
  
    public Clock(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
    }  
  
    public void setTime(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
    }  
  
    public void display() {  
        System.out.println(h + ":" + m + ":" + s);  
    }  
  
}  
  
public class TestClock {  
    public static void main(String args[]) {  
        Clock c1 = new Clock();  
        Clock c2 = new Clock(5, 10, 20);  
        c1.display();  
        c2.display();  
    }  
}
```



6.1.4. Phương thức khởi tạo

- ❑ Phương thức khởi tạo sao chép (copy constructor)
- ❑ Có tham số là một đối tượng cùng lớp
- ❑ Được sử dụng để tạo ra một đối tượng mới “giống” với đối tượng đã có

```
//hàm xây dựng sao chép của lớp Clock
public Clock(Clock c) {
    h = c.h;
    m = c.m;
    s = c.s;
}
```

```
Clock c1 = new Clock(5, 10, 20);
Clock c2 = new Clock(c1);
```



6.1.5. Nạp chồng phương thức (overload)

- ❑ Định nghĩa nhiều phương thức trùng tên cùng tồn tại trong một lớp.
- ❑ Các phương thức trùng tên phải khác nhau về tham số
 - ❑ Khác nhau về số lượng tham số
 - ❑ Khác nhau về kiểu tham số
 - ❑ Khác nhau về thứ tự các tham số
- ❑ Lưu ý
 - ❑ Không quan tâm đến kiểm tra về của phương thức và tên của tham số.



6.1.5. Nạp chồng phương thức (overload)

```
public class Diem2D {  
    private int x;  
    private int y;  
  
    public Diem2D()  
    {  
        x=0;  
        y=0;|  
    }  
    public Diem2D(int x, int y)  
    {  
        this.x=x;  
        this.setY(y);  
    }  
    public Diem2D(Diem2D d)  
    {  
        this.setX(d.getX());  
        this.setY(d.getY());  
    }  
}
```

```
class Point2D {  
    public void hienthi() {  
        System.out.println("(" + x + ", " + y + ")");  
    }  
  
    public void hienthi(String s) {  
        System.out.println(s + "(" + x + ", " + y + ")");  
    }  
}
```



6.1.6. Thành phần tĩnh (static members)

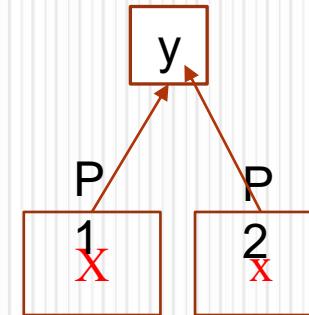
- ❑ Còn được gọi là thành phần “của lớp”.
- ❑ Khai báo:
 - ❑ Thêm từ khóa **static** sau thuộc tính truy xuất
 - ❑ Tồn tại độc lập với các đối tượng
 - ❑ Có bao nhiêu đối tượng được tạo ra thì vẫn chỉ có 1 bản (copy) của thành phần tĩnh trong bộ nhớ.
 - ❑ Truy xuất trực tiếp thông qua tên lớp
<Tên lớp>.<Tên thành phần tĩnh>
 - ❑ Các phương thức static chỉ được truy



6.1.6. Thành phần tĩnh (static members)

```
public class classA {  
    public int x;  
    public static int y;  
}
```

```
public static void main(String[] args) {  
    classA p1=new classA();  
    classA p2=new classA();  
}
```





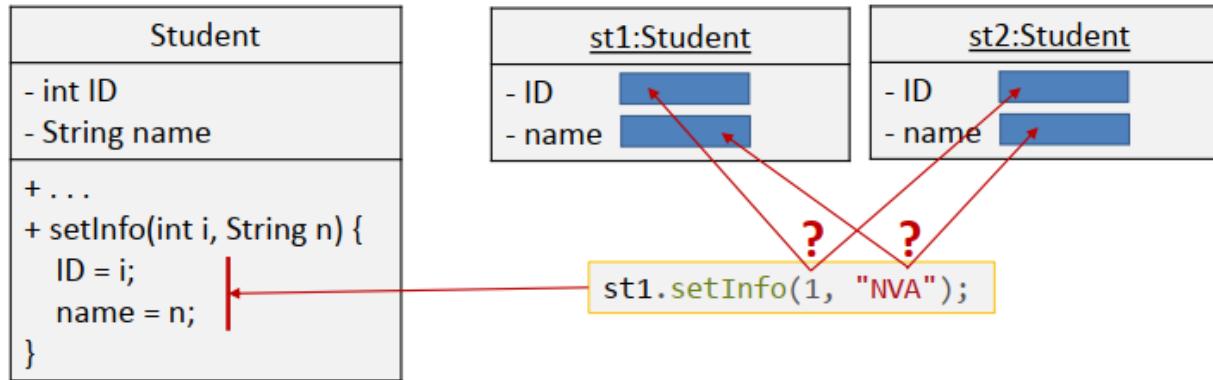
6.1.6. Thành phần tĩnh (static members)

```
class ClockStatic {  
    private static int count;  
    private int h, m, s;  
  
    public ClockStatic() {  
        h = m = s = 0;  
        count++;  
    }  
  
    public ClockStatic(int g, int p, int gi) {  
        h = g; m = p; s = gi;  
        count++;  
    }  
  
    static public void noOfInstances() {  
        System.out.println("# of instances: " +  
                           count);  
    }  
  
    public void display() {  
        System.out.println(h + ":" + m + ":" + s);  
    }  
}  
  
class TestClockStatic {  
    public static  
        ClockStatic.noOfInstances();  
  
    ClockStatic  
        ClockStatic.noOfInstances();  
    }  
}
```

6.1.7. Biến tham chiếu this

- ☐ Là một biến tham chiếu đặc biệt, tồn tại trong tất cả **các phương thức không tĩnh** của lớp.

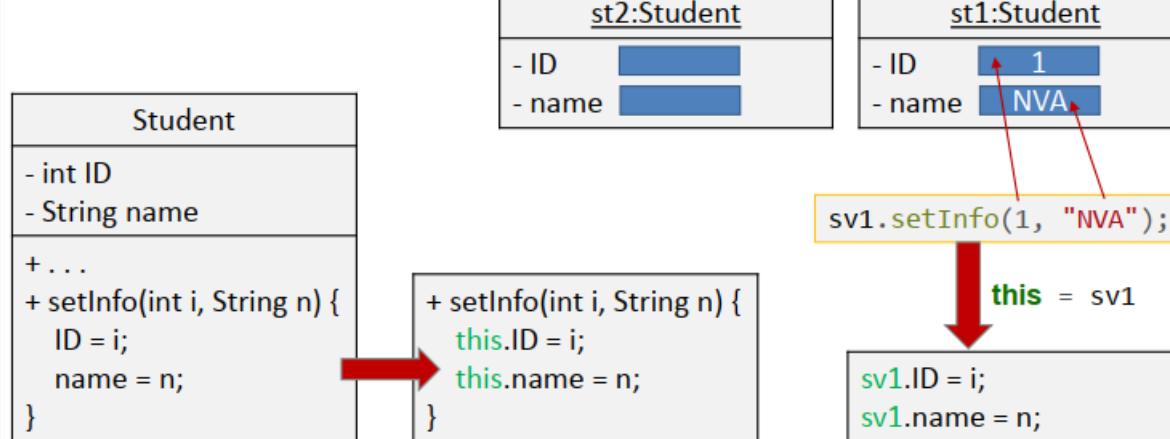
- ☐ **Điều này tham chiếu đến đối tượng đang**



6.1.7. Biến tham chiếu this

Trong tất cả các phương thức **không tĩnh**, tất cả các lệnh truy xuất vào các dữ liệu thành viên không tĩnh sẽ được tự

~~đưa ra~~ ~~đưa ra~~ ~~đưa ra~~ ~~đưa ra~~ ~~đưa ra~~



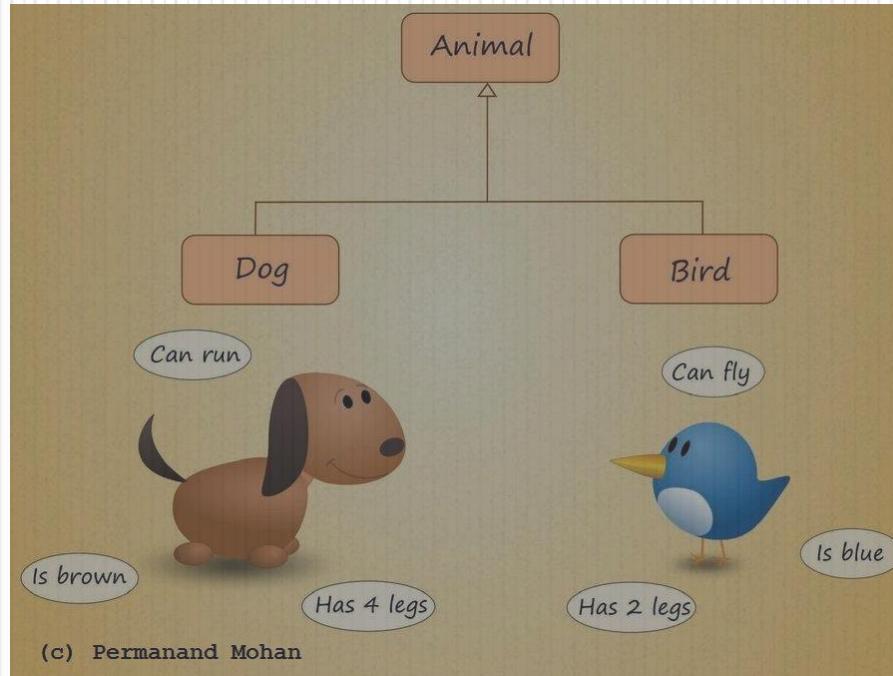


6.1.7. Biến tham chiếu this

❑ Biến tham chiếu this thường được sử dụng một cách tự động minh để tránh trùng

```
class Student {  
    private int ID;  
    private String name;  
  
    //...  
  
    public void setInfo(int ID, String name) {  
        this.ID = ID;  
        this.name = name;  
    }  
}  
  
public void setInfo(int ID, String name) {  
    ID = ID;  
    name = name;  
}  
}
```

6.2. Kế thừa (Inheritance)





6.2. Kế thừa (Inheritance)

- ❑ Kế thừa là quá trình mà các thuộc tính và hành vi được truyền từ thực thể cha đến thực thể con.
- ❑ Lớp kế thừa được gọi là lớp con (subclass) hay lớp dẫn xuất (devired class). Lớp cho phép lớp con kế thừa được gọi là lớp cha hay lớp cơ sở (super class).
- ❑ Lớp con thừa kế (có) các thành phần của lớp cha **ngoại trừ** các thành phần được khai báo truy cập là



6.2. Kế thừa (Inheritance)

- ❑ Lớp con cũng kế thừa các thành phần default của lớp cha nếu lớp con và lớp cha được định nghĩa trong cùng pakage.
- ❑ Lớp con không kế thừa constructor của lớp cha.
- ❑ Lớp con có thể nạp đè (overriding) các phương thức của lớp cha.
- ❑ Nếu một lớp không kế thừa từ lớp



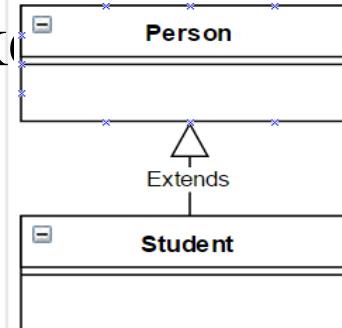
6.2.1. Tạo lớp kế thừa

❑ Cú pháp:

```
modifier class <subclass name> extends <superclass name> {  
    //subclass members  
}
```

❑ Ví dụ: Tạo lớp Student

```
public class Student extends Person {  
    //Các thành phần của Lớp Student  
}
```





6.2.2. Constructor trong kế thừa

- ❑ Khi một đối tượng thuộc lớp con được tạo ra:
 - ❑ Constructor tương ứng của lớp con sẽ được gọi.
 - ❑ Nếu constructor của lớp con không gọi constructor của lớp cha thì constructor mặc định của lớp cha sẽ tự động được gọi trước khi constructor của lớp con được thực thi.



6.2.2. Constructor trong kế thừa

- ❑ Khi một đối tượng thuộc lớp con được tạo ra:
 - ❑ Constructor tương ứng của lớp con sẽ được gọi.
 - ❑ Nếu constructor của lớp con không gọi constructor của lớp cha thì constructor mặc định của lớp cha sẽ tự động được gọi trước khi constructor của lớp con được thực thi.
- ❑ Từ khóa **super** được sử dụng để gọi constructor của lớp cha:
 - ❑ **super**([các tham số có constructor của lớp cha]);



6.2.2. Constructor trong kế thừa

```
public class GiaoVien {
    String Hoten;
    double HSLuongGV;
    int SoNgayNghi;
    public GiaoVien(){
        Hoten="Nguyen Van A";
        HSLuongGV=2.34;
        SoNgayNghi=0;
    }
    public GiaoVien(String ten, double luong, int sn){
        Hoten=ten;
        HSLuongGV=luong;
        SoNgayNghi=sn;
    }
    public double TinhLuong(){
        return HSLuongGV*1300;
    }
    public void GiangDay(){
    }
}

public class GVCN extends GiaoVien{
    String LopCN;
    public GVCN(String ten, double luong, int sn, String lop) {
        super(ten,luong,sn);
        LopCN=lop;
    }
    public void SinhHoatCN() {
    }
}
```



6.2.3. Các dạng kế thừa

Single Inheritance	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance	<pre>public class A { } public class B extends A { } public class C extends B { }</pre>
Hierarchical Inheritance	<pre>public class A { } public class B extends A { } public class C extends A { }</pre>
Multiple Inheritance	<pre>public class A { } public class B { } public class C extends A,B { } // Java does not support multiple Inheritance</pre>



6.2.4. Ghi đè (Override)

- Lớp con có thể có thành viên (*thuộc tính/phương thức*) trùng với thành viên của lớp cha.
- Lớp con định nghĩa một phương thức có **chữ ký** (*tên + tham số*) trùng với phương thức thành viên của lớp cha gọi là override (*nạp đè/ghi đè phương thức*).
- Khi một phương thức của lớp cha bị ghi đè, nó sẽ **bị “che”** đi bởi phương thức của lớp con.
- Muốn gọi phương thức bị che ở lớp cha, sử dụng tham chiếu **super**.



6.2.4. Nạp đè (Override)

The screenshot shows a Java code editor window titled "TestDog.java" and a terminal window showing the execution and output of the code.

TestDog.java:

```
1  class Animal {
2      public void move() {
3          System.out.println("Animals can move");
4      }
5  }
6
7  class Dog extends Animal {
8      public void move() {
9          System.out.println("Dogs can walk and run");
10     }
11 }
12
13 public class TestDog {
14
15     public static void main(String args[]) {
16         Animal a = new Animal(); // Animal reference and object
17         Animal b = new Dog(); // Animal reference but Dog object
18
19         a.move(); // runs the method in Animal class
20         b.move(); // runs the method in Dog class
21     }
22 }
```

Terminal Output:

```
C:\WINDOWS\system32\cmd.exe
E:\java>javac TestDog.java
E:\java>java TestDog
Animals can move
Dogs can walk and run
E:\java>
```



6.2.4. Nạp đè (Override)

```
TestDog3.java |  
1  class Animal {  
2      public void move() {  
3          System.out.println("Animals can move");  
4      }  
5  }  
6  
7  class Dog extends Animal {  
8      public void move() {  
9          super.move(); // invokes the super class method  
10         System.out.println("Dogs can walk and run");  
11     }  
12 }  
13  
14 public class TestDog3 {  
15  
16     public static void main(String args[]) {  
17         Animal b = new Dog(); // Animal reference but Dog object  
18         b.move(); // runs the method in Dog class  
19     }  
20 }
```

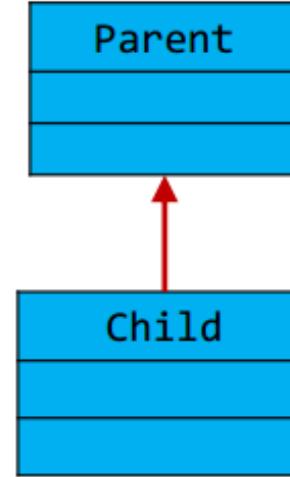
C:\WINDOWS\system32\cmd.exe
E:\java>javac TestDog3.java
E:\java>java TestDog3
Animals can move
Dogs can walk and run
E:\java>



Sự tương thích giữa tham chiếu và đối tượng

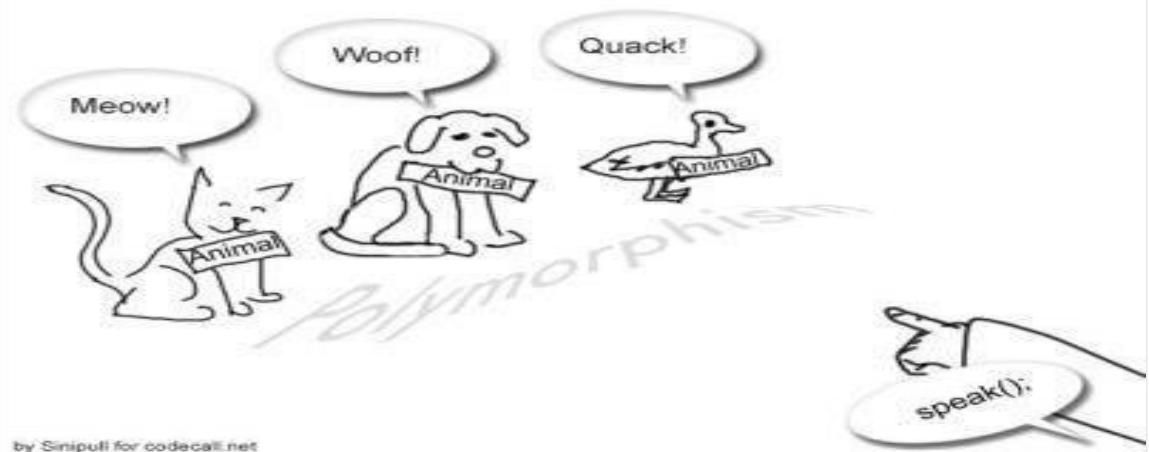
- ❑ Tham chiếu thuộc lớp cha có thể tham chiếu đến:
 - ❑ Đối tượng thuộc lớp cha
 - ❑ Đối tượng thuộc lớp con
- Một tham chiếu thuộc lớp con chỉ có thể tham chiếu đến đối tượng thuộc lớp con.

```
Parent p;  
p = new Parent(...); ✓  
p = new Child(...); ✓  
  
Child c;  
c = new Child(...); ✓  
c = new Parent(...); ✗
```



6.3. Đa hình

- ✓ Cùng một thông điệp nhưng sẽ được xử lý khác nhau tùy vào ngữ cảnh cụ thể.
- ✓ Cùng yêu cầu nhưng mỗi đối tượng có đáp ứng khác nhau.
- ✓ Chỉ được thể hiện khi có sử dụng **kế thừa + override phương thức**.





6.3. Đa hình

```
public class Animal {  
    public void eat() {  
        System.out.println("Eating...");  
    }  
}  
  
public class BabyDog extends Animal{  
    public void eat() {  
        System.out.println("Drinking milk...");  
    }  
}  
  
public class MonKey extends Animal{  
    public void eat() {  
        System.out.println("Eating fruits...");  
    }  
}  
  
public class AnimalDemo {  
    public static void main(String[] args) {  
        Animal a[] = new Animal[3];  
        a[0] = new Animal();  
        a[1] = new MonKey();  
        a[2] = new BabyDog();  
        for (int i = 0; i < 3; i++)  
            a[i].eat();  
    }  
}
```

Eating...
Eating fruits...
Drinking milk...



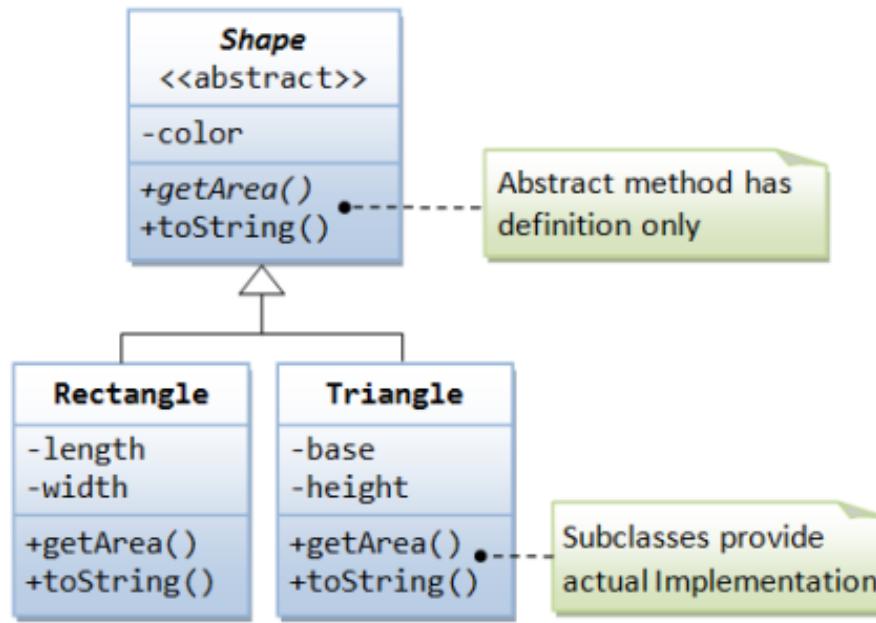
6.4. Phương thức trừu tượng và lớp trừu tượng

- ❑ Là phương thức **chỉ có khai báo**, không có cài đặt.
- ❑ Dùng từ khóa **abstract** được dùng để khai báo một phương thức là trừu tượng.
- ❑ Các lớp con phải cài đặt các phương thức trừu tượng của lớp cha.
- ❑ Dùng cho các phương thức chưa có định nghĩa cụ thể trong ngữ cảnh của lớp đó.
- ❑ Là một phương pháp để bắt buộc các lớp con phải cài đặt các phương thức theo yêu cầu.



6.4. Phương thức trừu tượng và lớp trừu tượng

```
public class Shape {  
    //các thành viên khác ...  
  
    public abstract double getArea();      //diện tích các Loại hình khác  
                                         //nhau thì khác nhau về cách tính  
}  
}
```





6.4. Phương thức trừu tượng và lớp trừu tượng

```
public class TestShape {  
    public static void main(String[] args) { Shape s1  
        = new Rectangle("red", 4, 5);  
        System.out.println(s1);  
        System.out.println("Area is " + s1.getArea());  
  
        Shape s2 = new Triangle("blue", 4, 5);  
        System.out.println(s2);  
        System.out.println("Area is " + s2.getArea());  
  
        // Cannot create instance of an abstract class  
        Shape s3 = new Shape("green"); //Compilation Error!!  
    }  
}
```



6.5. Interface

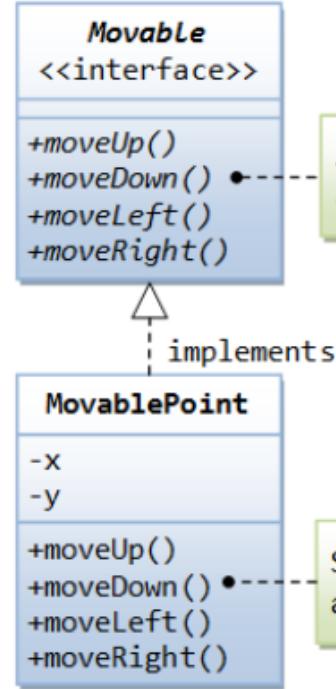
- ❑ Interface (Giao diện) có thể xem như là một lớp hoàn toàn ảo: tất cả phương thức đều không được cài đặt.
- ❑ Một interface chỉ chứa các khai báo các phương thức có phạm vi truy cập là public hoặc các hằng số (public static final...).
- ❑ Cú pháp

```
[public] interface <interfaceName> [extends <superInterface>] {  
    //khai báo của các hằng số (static final ...)  
  
    //khai báo các phương thức  
}
```



6.5. Interface

```
public interface Movable {  
    //abstract methods to be implemented  
    //by the subclasses  
    public void moveUp();  
    public void moveDown();  
    public void moveLeft();  
    public void moveRight();  
}
```



Abstract method has definition only

Subclasses provide actual implementation



6.5. Interface

```
public class MovablePoint implements Movable {
    private int x, y;      //coordinates of the point

    public MovablePoint(int x, int y) {
        this.x = x;      this.y = y;
    }
    public String toString() {
        return "(" + x + "," + y + ")";
    }
    public void moveUp() { y--; }
    public void moveDown() {
        y++;
    }
    public void moveLeft() {
        x--;
    }
    public void moveRight() {
        x++;
    }
}

public class TestMovable {
    public static void main(String[] args) {
        Movable m1 = new MovablePoint(5, 5);

        System.out.println(m1);
        m1.moveDown();
        System.out.println(m1);
        m1.moveRight();
        System.out.println(m1);
    }
}
```

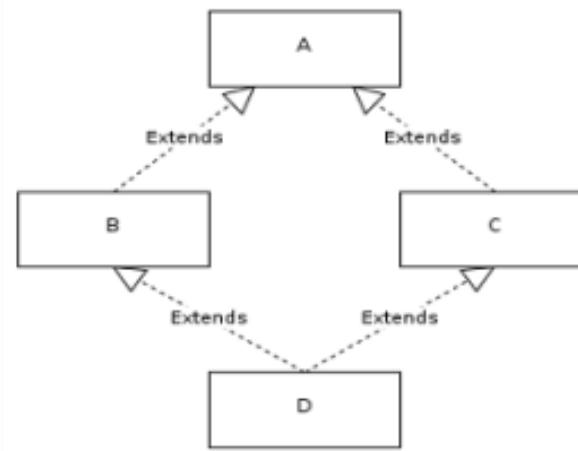
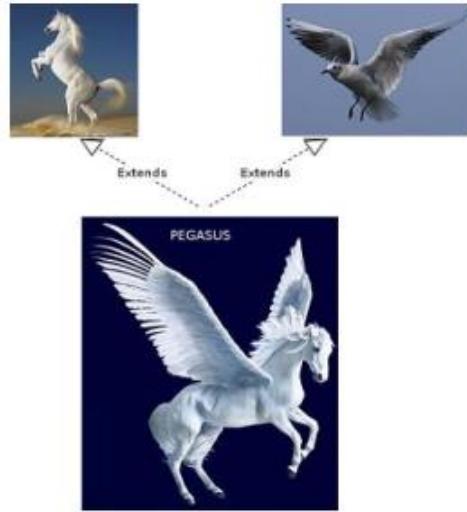


6.5. Interface

❑ Phương thức mặc định (default)

- ❑ Từ Java 8, các interface có thể có các phương thức mặc định, là phương thức được cài đặt.
- ❑ Các lớp hiện thực (implement) interface có thể cài đặt hoặc không cài đặt các phương thức mặc định.
 - ❑ Không cài đặt: kế thừa phương thức mặc định.
 - ❑ Cài đặt: overriding phương thức mặc định.
- ❑ Cú pháp: thêm từ khóa **default** trước khai báo.

6.5. Đa thừa kế



- ❑ Một lớp con được thừa kế từ nhiều lớp cha.
- ❑ Java không hỗ trợ đa thừa kế.



6.5. Đa thừa kế

❑ Java hỗ trợ đa thừa kế kiểu:

❑ Một lớp có thể hiện thực nhiều

```
//One class implements multiple interfaces
public class <classname> implements <interface names> {
    ...
}
```

```
public class <classname> extends <superclass> implements <interfaces> {
    ...
}
```

hiện thực interface



Giả lập đa kế thừa

- ❑ Dùng phương thức **default** của interface
 - ❑ Chỉ kế thừa được phương thức
 - ❑ Chỉ được hỗ trợ từ Java 8

```
public class Button implements Clickable, Accessible {  
    public static void main(String[] args) {  
        Button button = new Button();  
        button.click();  
        button.access();  
    }  
}
```

```
interface Clickable{  
    default void click(){  
        System.out.println("click");  
    }  
}  
  
interface Accessible{  
    default void access(){  
        System.out.println("access");  
    }  
}
```



Giả lập đa kế thừa

- ❑ Dùng quan hệ composition (bao gồm):
 - ❑ Không chính xác về ngữ nghĩa
 - ❑ Cho phép kế thừa cả thuộc tính và phương thức.



7. Xử lý ngoại lệ

7.1. Cơ bản về ngoại lệ

7.2. Xử lý ngoại lệ bằng lệnh
try...catch...finally

7.3. Lệnh throw và throws

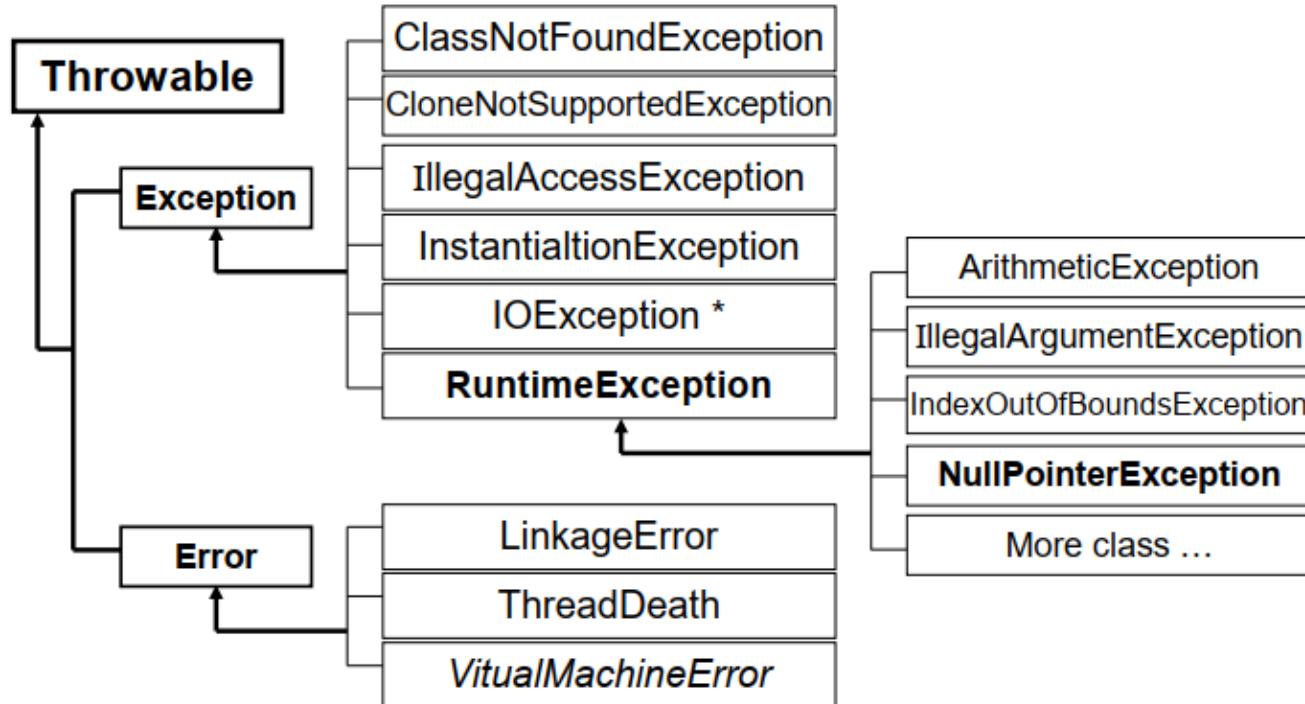


7.1. Cơ bản về ngoại lệ (Exception)

- ❑ Ngoại lệ là những lỗi đặc biệt có thể phát sinh trong quá trình thực hiện chương trình và làm chương trình bị ngắt đột ngột.
- ❑ Ví dụ về một số nguyên nhân làm xuất hiện exception:
 - ❑ Nhập dữ liệu không hợp lệ;
 - ❑ Không tìm thấy file;
 - ❑ Kết nối mạng không tìm thấy hoặc JVM tràn bộ nhớ...



7.1. Ngoại lệ (Exception)





7.1. Ngoại lệ (Exception)

□ Phân loại Exception

□ **Checked Exception:** xảy ra tại thời điểm compile time, lỗi do người dùng mà không thể lường trước được bởi lập trình viên. Ví dụ: đọc một file nhưng file đó không được tìm thấy... → Không cần “catch”

□ **Unchecked Exception:** xảy ra ở thời điểm runtime, ngoại lệ này thường liên quan đến lỗi logic, là ngoại lệ có thể tránh được bởi lập trình viên. Ví dụ: lấy x/y ($y = 0$), truy cập đến phần tử vượt quá chiều dài của mảng...



7.2. Xử lý ngoại lệ

❑ Xử lý lỗi truyền thông

- ❑ Cài đặt mã xử lý lỗi tại nơi phát sinh ra lỗi:
 - ❑ Làm cho chương trình trở nên rối rắm, khó hiểu;
 - ❑ Không phải lúc nào cũng có đầy đủ thông tin để xử lý;
 - ❑ Không nhất thiết phải xử lý tại lúc đó...
- ❑ Khó kiểm soát được hết các trường hợp như lỗi số học, lỗi bộ nhớ ...
- ❑ Lập trình viên có thể quên xử lý lỗi.



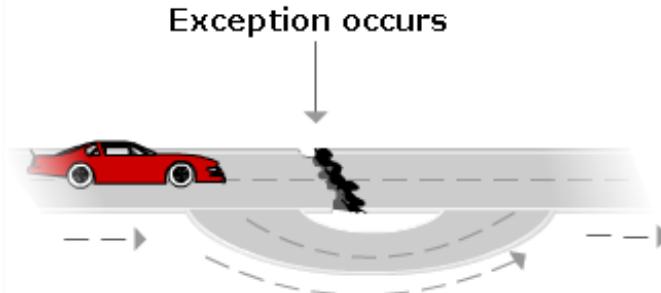
7.2. Xử lý ngoại lệ

❑ Xử lý ngoại lệ

- ❑ Sử dụng khối try – catch
- ❑ Sử dụng nhiều khối catch
- ❑ Sử dụng try – catch - finally
- ❑ Sử dụng từ khóa throw, throws



7.2.1. Sử dụng try - catch



Syntax

```
try {
    Statement_1;
    Statement_2;
    ...
}
catch (ExceptionType
        ObjectName) {
    St1;
}
```

try-catch

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Scanner sc = new Scanner(System.in);
    int kq = 0;

    try {
        System.out.println("Nhập x = ");
        int x = sc.nextInt();
        System.out.println("Nhập y = ");
        int y = sc.nextInt();
        kq = x / y;
    } catch (Exception e) {
        System.out.println("Lỗi:" + e);
    }
    System.out.println(" x/y = " + kq);
    sc.close();
}
```



7.2.1. Sử dụng try - catch

- ☐ Sử dụng nhiều khối catch

Syntax

```
try{  
    Statement_1;  
    Statement_2;  
    ...  
} catch (ExceptionType name1) {  
    st1;  
} catch (ExceptionType name2) {  
    st2;  
}  
...  
...
```

```
try {  
    System.out.println("Nhập x = ");  
    int x = sc.nextInt();  
    System.out.println("Nhập y = ");  
    int y = sc.nextInt();  
    kq = x / y;  
    System.out.println(" x/y = " + kq);  
  
    int [] a = {1, 2, 3, 4, 5, 6};  
    System.out.println("Nhập i = ");  
    int i = sc.nextInt();  
    System.out.println("a[i] = "+a[i]);  
} catch (ArithmException e){  
    System.out.println("Lỗi: " + e);  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Chi số vượt quá giới hạn!");  
} catch (Exception e){  
    System.out.println("Exception gốc!");  
}
```



7.2.2. Sử dụng try – catch - finally

Syntax

```
try {
    Statement_1;
    Statement_2;
}
catch (ExceptionType ObjectName) {
    Statement_i;
}
finally{
    //Clean up code
    Statement_j;
}
```

```
Scanner sc = new Scanner(System.in);
int kq = 0;

try {
    System.out.println("Nhập x = ");
    int x = sc.nextInt();
    System.out.println("Nhập y = ");
    int y = sc.nextInt();
    kq = x / y;
    System.out.println(" x/y = " + kq);

    int [] a = {1, 2, 3, 4, 5, 6};
    System.out.println("Nhập i = ");
    int i = sc.nextInt();
    System.out.println("a[i] = "+a[i]);
} catch (ArithmetricException e){
    System.out.println("Lỗi: " + e);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Chi số vượt quá giới hạn!");
} catch (Exception e){
    System.out.println("Exception gốc !");
} finally {
    System.out.println("Câu lệnh này luôn thực hiện !");
}
sc.close();
```

Khối **finally** sẽ luôn luôn được thực thi dù có xảy ra hay không xảy ra ngoại lệ.



7.2.2. Sử dụng try – catch - finally

```
try {
    System.out.print("Nhập x = ");
    int x = sc.nextInt();
    System.out.print("Nhập y = ");
    int y = sc.nextInt();
    kq = x/y;
    System.out.println("x/y = " + kq);
    int[] a= {1,2,3,4,5,6};
    System.out.print("Nhập i = ");
    int i = sc.nextInt();
    System.out.println("a["+ i +"]=" + a[i]);
}
catch (ArithmetricException e) {
    System.out.println("Lỗi số học" + e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Chỉ số vượt giới hạn" + e.getMessage());
}
catch (Exception e) {
    System.out.println("Lỗi rồi nhé!" + e.getMessage());
}
finally {
    System.out.println("Kết thúc chương trình");
}
```

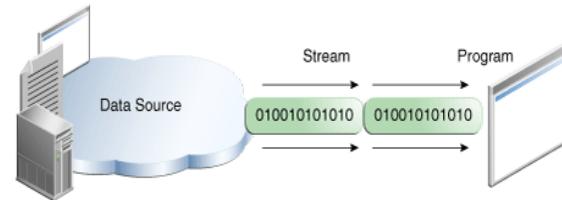
Nhập x = 4
Nhập y = 0
Lỗi số học/ by zero
Kết thúc chương trình
BUILD SUCCESSFUL (total time: 2 seconds)

Nhập x = 8
Nhập y = 4
x/y = 2
Nhập i = 7
Chỉ số vượt giới hạn 7
Kết thúc chương trình
BUILD SUCCESSFUL (total time: 6 seconds)

Nhập x = 8
Nhập y = 4
x/y = 2
Nhập i = 3
a[3]=4
Kết thúc chương trình
BUILD SUCCESSFUL (total time: 5 seconds)

8. Các luồng nhập xuất dữ liệu

- ❑ Giới thiệu về nhập xuất trong Java
- ❑ Lớp java.io.File
- ❑ Luồng nhập xuất (Stream)
- ❑ Luồng nhập xuất theo byte
- ❑ Luồng nhập xuất theo ký tự
- ❑ Nhập xuất đối tượng





8.1. Giới thiệu về nhập xuất trong Java

- ❑ Các gói hỗ trợ nhập xuất trong JDK
 - ❑ Java.io: nhập xuất chuẩn (standart I/O)
 - ❑ Được giới thiệu từ JDK 1.0
 - ❑ Nhập xuất thông qua Stream
 - ❑ Java.nio: nhập xuất mới (new I/O)
 - ❑ Được giới thiệu từ JDK 1.4
 - ❑ Nâng cao hiệu quả việc nhập xuất qua vùng đệm
 - ❑ Được giới thiệu từ JDK 1.7



8.2. Lớp java.io.File

- ❑ Đối tượng **File** biểu diễn 1 tập tin hoặc 1 thư mục.
- ❑ Khởi tạo 1 đối tượng
`public File(String pathString)`
- ❑ Sử dụng đường dẫn (path) theo dạng:
 - ❑ Trong Windows: “D:\\ViduJava\\Hello.java”
 - ❑ Trong Unix/Mac: “/ViduJava/Hello.java”
- ❑ Ví dụ
 - ❑ `File f1 = new File("data.txt")`
 - ❑ `File f2 = new File("C:\\\\ViDu\\\\Hello.java");`
 - ❑ `File dir1= new File("C:\\\\temp");`



8.2. Lớp java.io.File

❑ Một số phương thức thông dụng

- ❑ **public boolean exists();** // Có tồn tại hay không
- ❑ **public long length();** // Kích thước file
- ❑ **public boolean isDirectory();** // Là thư mục?
- ❑ **public boolean isFile();** // Là tập tin?
- ❑ **public boolean canRead();** // Có thể đọc?
- ❑ **public boolean canWrite();** // Có thể ghi?
- ❑ **public boolean delete();** // Xóa
- ❑ **public void deleteOnExit();** // Xóa khi kết thúc
- ❑ **public boolean renameTo(File dest);** // Đổi tên
- ❑ **public boolean mkdir();** // Tạo thư mục
- ❑ **public File[] listFiles();** // Liệt kê thư mục dạng File



8.2. Lớp java.io.File

- ❑ Ví dụ: truy xuất thông tin của một file

```
1 package javaio;
2
3 import java.io.File;
4
5 public class GetFileInfor {
6     public static void main(String[] args) {
7         File f = new File("D:\\TestJavaIO\\ABC.txt");
8         if (f.isDirectory())
9             System.out.println("Is folder");
10        if (f.isFile())
11            System.out.println("Is file");
12        if (f.exists())
13            System.out.println("File is exists!");
14        else
15            System.out.println("File does not exists");
16        System.out.println("Absolute path: " + f.getAbsolutePath());
17        System.out.println("Get path: " + f.getPath());
18        System.out.println("Get name: " + f.getName());
19        System.out.println("Get parent: " + f.getParent());
20    }
21 }
```



8.2. Lớp java.io.File

- ❑ Ví dụ: hiển thi danh sách file trong một

```
1  package javaio;
2
3  import java.io.File;
4
5  public class ListFilesInFolder {
6      public static void main(String[] args) {
7          File f = new File("C:\\windows");
8          if (f.isDirectory()){
9              File [] list = f.listFiles();
10             for (File item: list){
11                 System.out.println(item.getPath());
12             }
13         }
14     }
15 }
```



8.3. Lớp java.nio.Files

❑ Files

❑ import java.nio.Files

❑ Một số phương thức thông dụng:

❑ static **List<String>** **readAllLines**(Path p, Charset cs)

Đọc tất cả các dòng của một tập tin → mảng các dòng (List).

❑ static **byte[]** **readAllBytes**(Path p)

Đọc toàn bộ nội dung của tập tin → mảng các byte.



8.3. Lớp java.nio.Files

❑ Ví dụ: Đọc toàn bộ nội dung tập tin

```
1 package javaio;
2
3 import java.io.IOException;
4 import java.nio.charset.StandardCharsets;
5 import java.nio.file.Files;
6 import java.nio.file.Paths;
7
8 public class ReadContentFile2 {
9     public static void main(String[] args) throws IOException {
10         byte[] encoded = Files.readAllBytes(Paths.get("D:\\TestJavaIO\\ABC.txt"));
11         String s = new String(encoded, StandardCharsets.UTF_8);
12         System.out.println(s);
13     }
14 }
```



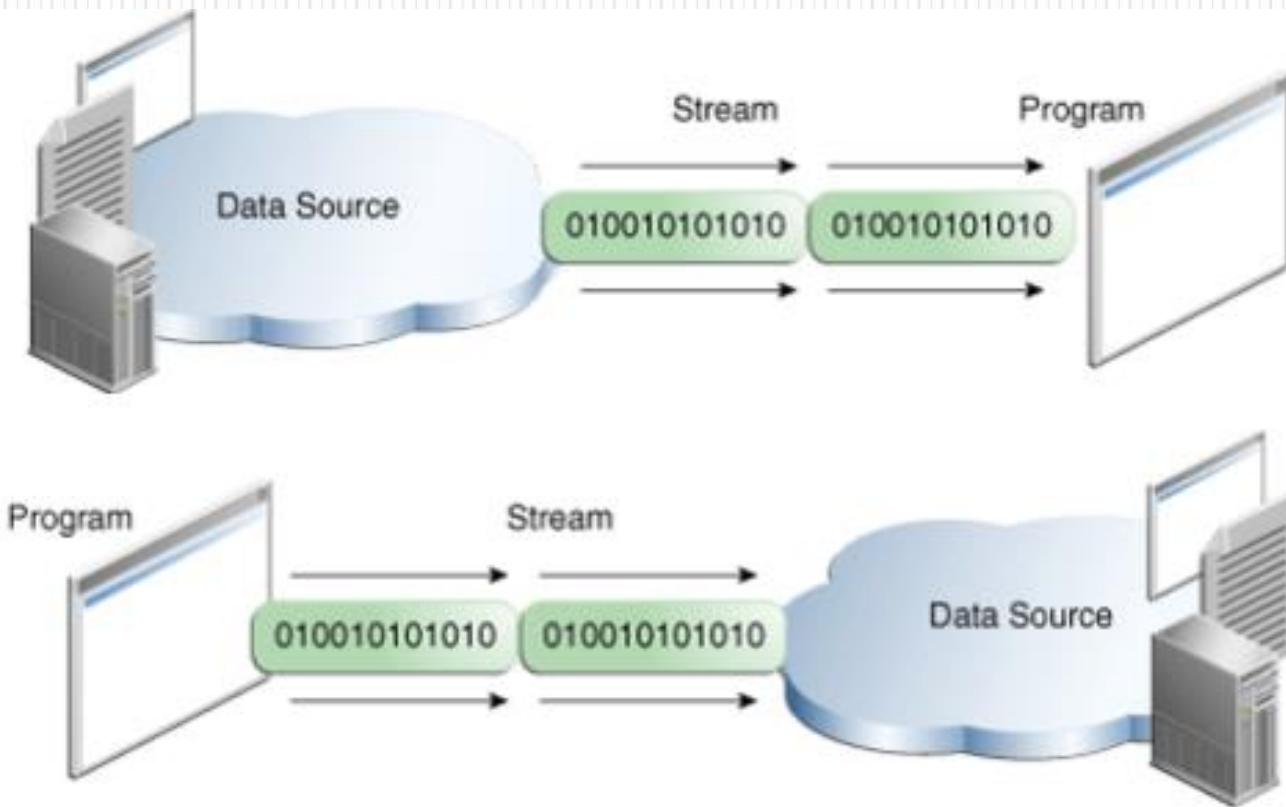
8.3. Lớp java.nio.Files

- ❑ Ví dụ: Đọc toàn bộ nội dung tập tin

```
1 package javaio;
2
3 import java.io.IOException;
4 import java.nio.charset.StandardCharsets;
5 import java.nio.file.Files;
6 import java.nio.file.Paths;
7 import java.util.List;
8
9 public class ReadContentFile {
10     public static void main(String[] args) throws IOException {
11         List<String> list = Files.readAllLines(
12             Paths.get("D:\\TestJavaIO\\ABC.txt"), StandardCharsets.UTF_8);
13         for (String s: list){
14             System.out.println(s);
15         }
16     }
17 }
```

8.4. Khái niệm về luồng (Stream)

Stream là một dòng tuần tự các byte, có thứ tự (*chỉ đi theo 1 chiều*) dùng để truyền tải dữ liệu giữa chương trình và nguồn dữ liệu (các thiết bị)





8.4. Khái niệm về luồng (Stream)

❑ FileOutputStream

- ❑ Sử dụng để ghi dữ liệu ra tập tin
- ❑ Khởi tạo:
 - ❑ FileOutputStream(File file)
 - ❑ FileOutputStream(String name)

❑ FileInputStream

- ❑ Sử dụng để truy xuất dữ liệu từ tập tin
- ❑ Khởi tạo
 - ❑ FileInputStream(File file)
 - ❑ FileInputStream(String name)



8.5. Các bước đọc/ghi tập tin

❑ **Ghi**

- ❑ Tạo đối tượng **FileOutputStream** từ một đường dẫn tập tin hoặc một đối tượng File.
- ❑ Tạo đối tượng **Writer/OutputStream** kết nối với đối tượng **FileOutputStream**.
- ❑ Sử dụng đối tượng **Writer/OutputStream** để ghi dữ liệu

❑ **Đọc**

- ❑ Tạo đối tượng **FileInputStream** từ một đường dẫn tập tin hoặc một đối tượng File.
- ❑ Tạo đối tượng **Reader/InputStream** kết nối với đối tượng **FileInputStream**.
- ❑ Sử dụng đối tượng **Reader/InputStream** để đọc dữ liệu.



8.5.1. Character Stream – Ghi file

```
public class WriteFileCharacterStream {  
    public static void main(String[] args) {  
        try{  
            FileOutputStream fis = new FileOutputStream("D:\\TestJavaIO\\ABC.txt");  
            OutputStreamWriter osw = new OutputStreamWriter(fis,"UTF-8");  
            BufferedWriter bw = new BufferedWriter(osw);  
            String[] str = new String[]{"ĐH CNTP TPHCM", "Khoa CNTT", "BM HTTT"};  
            for (String s: str){  
                bw.write(s);  
                bw.write("\n");  
            }  
            bw.close();  
        }  
        catch(Exception ex){  
            ex.printStackTrace();  
        }  
    }  
}
```



8.5.2. Character Stream – Đọc file

```
public class ReadFileCharacterStream {
    public static void main(String[] args) {
        try{
            FileInputStream fis = new FileInputStream("D:\\TestJavaIO\\ABC.txt");
            InputStreamReader isr = new InputStreamReader(fis, "UTF-8");
            BufferedReader br = new BufferedReader(isr);
            String s;
            do{
                s = null;
                s = br.readLine();
                System.out.println(s);
            }
            while (s != null);
            br.close();
        }
        catch(Exception ex){
            ex.printStackTrace();
        }
    }
}
```



8.5.3. Bye Stream – Ghi đối tượng ra file

```
1 package bytestream;
2
3 import java.io.Serializable;
4
5 public class PhanSo implements Serializable{
6     private int tuso;
7     private int mauso;
8     public PhanSo(int t, int m) {
9         this.tuso = t;
10        this.mauso = m;
11    }
12    PhanSo() {
13        this.tuso = 0;
14        this.mauso = 1;
15    }
16    public void xuat(){
17        System.out.println(this.tuso + "/" + this.mauso);
18    }
19 }
```



8.5.3. Bye Stream – Ghi đối tượng ra file

```
public class WriteFileByByteStream {  
    public static void main(String[] args) {  
        try{  
            FileOutputStream fos = new FileOutputStream("D:\\TestJavaIO\\PhanSo.obj");  
            ObjectOutputStream oos = new ObjectOutputStream(fos);  
            oos.writeChar(100);  
            oos.writeObject(new PhanSo(1,3));  
            oos.writeObject(new PhanSo(2,5));  
            oos.writeObject(new PhanSo(3,4));  
            oos.writeObject(new PhanSo(4,5));  
            oos.close();  
        }  
        catch(Exception ex){  
            ex.printStackTrace();  
        }  
    }  
}
```



8.5.3. Bye Stream – đọc đối tượng từ file

```
public class ReadFileByteStream {
    public static void main(String[] args) {
        try{
            FileInputStream fis = new FileInputStream("D:\\TestJavaIO\\PhanSo.obj");
            ObjectInputStream ois = new ObjectInputStream(fis);
            int t = ois.readChar();
            System.out.println(t);
            PhanSo p = (PhanSo)ois.readObject();
            p.xuat();
            p = (PhanSo)ois.readObject();
            p.xuat();
            p = (PhanSo)ois.readObject();
            p.xuat();
            p = (PhanSo)ois.readObject();
            p.xuat();
            ois.close();
        }
        catch(Exception ex){
            ex.printStackTrace();
        }
    }
}
```