

## JAVA CORE CHO MỌI NGƯỜI

HƯỚNG DẪN SỬ DỤNG .....	6
<b>1 PHẦN 1 KHÁI NIỆM LẬP TRÌNH CHO NGƯỜI MỚI BẮT ĐẦU .....</b>	<b>8</b>
1.1 Chương trình phần mềm là gì.....	8
1.2 Lập trình là gì.....	8
1.3 Ngôn ngữ lập trình là gì.....	8
1.4 Các phương pháp lập trình.....	8
1.4.1 Phương pháp lập trình tuần tự (Sequential Programming) .....	8
1.4.2 Phương pháp lập trình cấu trúc (Structured Programming) .....	9
1.4.3 Phương pháp lập trình thủ tục(Procedural Programming) .....	9
1.4.4 Phương pháp lập trình hướng đối tượng (Object-Oriented Programming) .....	10
1.5 Lập trình ứng dụng máy tính (Destop APP) .....	10
1.6 Lập trình ứng dụng di động .....	10
1.7 Lập trình web.....	11
1.8 Lập trình core service.....	11
1.9 Lập trình nhúng .....	12
1.10 Công cụ soạn thảo code(IDE) phổ biến .....	12
1.11 Các thuật ngữ sử dụng trong lập trình .....	12
1.11.1 Ngôn ngữ lập trình bậc thấp .....	12
1.11.2 Ngôn ngữ lập trình bậc cao.....	13
1.11.3 Mã nguồn .....	13
1.11.4 Mã máy.....	13
1.11.5 Trình Biên dịch chương trình phần mềm .....	14
1.11.6 Trình Thông dịch chương trình phần mềm .....	14
1.11.7 Bug và Debug (lỗi phần mềm và Gỡ lỗi).....	14
1.11.8 Exception (Ngoại lệ) .....	15
1.11.9 Console.....	15
1.11.10 GUI (Graphical User Interface).....	15
1.11.11 Database.....	16
1.11.12 Var (Biến).....	16
1.11.13 Funtion, Procedure, Method (Hàm, Thủ tục, Phương thức) .....	17

1.11.14	Giải thuật.....	17
1.11.15	Argument (Đối số).....	18
1.11.16	Parameter (Tham số).....	18
1.11.17	Test (Kiểm thử phần mềm).....	19
2	PHẦN 2 JAVA BASIC DÀNH CHO NGƯỜI MỚI BẮT ĐẦU .....	19
2.1	Tại sao chúng ta lại nên lập trình bằng Java .....	19
2.2	Tìm hiểu đặc điểm ngôn ngữ lập trình java trước khi lập trình Java .....	20
2.2.1	Ngôn ngữ Java đơn giản dễ dùng .....	21
2.2.2	Ngôn ngữ Java Độc lập nền tảng .....	21
2.2.3	Java hướng đối tượng hoàn toàn .....	22
2.2.4	Lập trình xử lý đa luồng (Multi-thread) .....	23
2.2.5	Hiệu năng cao (High performance) .....	23
2.2.6	Mạnh mẽ (Robust) .....	23
2.3	Tìm hiểu về Kiến trúc Java.....	23
2.4	Cài đặt môi trường để lập trình Java.....	24
2.4.1	Cài đặt JDK lên máy phát triển.....	24
2.4.2	Tìm kiếm một bộ soạn thảo code phù hợp nhất.....	25
2.5	Lập trình chương trình Java đầu tay.....	29
2.5.1	Tìm hiểu qua về chức năng bộ JDK một chút.....	29
2.5.2	Tìm hiểu về các ứng dụng Java.....	29
2.5.3	Tạo ứng dụng trên IDE.....	30
2.6	Tìm hiểu nhanh các Khái niệm cần biết đầu tiên trước khi lập trình Java .....	38
2.7	Chú ý các quy tắc trong viết code Java.....	40
2.7.1	Quy tắc đặt tên trong Java.....	40
2.7.2	Comment(Ghi chú) code trong Java .....	43
2.7.3	Các từ khóa quan trọng trong Java.....	47
2.8	Variables (Biến số) và Kiểu dữ liệu trong Java .....	51
2.8.1	Biến và phạm vi biến trong Java.....	51
2.8.2	Các kiểu dữ liệu nguyên thủy trong Java .....	55
2.8.3	Giá trị mặc định của biến có kiểu nguyên thủy .....	57
2.8.4	Các phép toán với các biến nguyên thủy kiểu số (Numeric).....	57
2.8.5	Chữ viết biểu thị giá trị trong Java ( <b>Literals</b> ).....	58
2.8.6	Chuyển đổi kiểu dữ liệu số trong Java (Casting).....	61
2.9	Kiểu dữ liệu char.....	63

2.10	Toán tử trong Java .....	65
2.10.1	Toán tử số học .....	65
2.10.2	Toán tử logic .....	66
2.11	Kiểu dữ liệu mảng trong Java .....	69
2.11.1	Mảng một chiều trong Java .....	70
2.11.2	Mảng hai chiều trong Java .....	73
2.12	Cấu trúc điều khiển trong Java .....	74
2.12.1	Cấu trúc điều kiện rẽ nhánh if...else trong Java .....	75
2.12.2	Cấu trúc điều kiện rẽ nhánh switch-case trong Java .....	76
2.12.3	Cấu trúc Lặp While và do-while trong Java .....	78
2.12.4	Cấu trúc lặp for... trong Java .....	80
2.13	Nghiên cứu sâu về <b>Classes</b> trong Java.....	86
2.13.1	Package và cách tổ chức thư mục Trong Java.....	86
2.13.2	Khai báo Class (Declaring Classes) trong Java .....	89
2.13.3	Khai báo biến thành viên (Declaring Member Variables) trong Java.....	91
2.13.4	Khai báo phương thức (Defining Methods) trong Java.....	93
2.13.5	Phương thức khởi dựng (contructor method) .....	96
2.13.6	Đối tượng(Object) là gì, phân biệt giữa đối tượng (Object) và Class trong Java.....	97
2.13.7	Từ khóa this là gì và sử dụng thế nào .....	98
2.14	Access Modifiers Trong Java.....	99
2.15	Lớp kế thừa trong Java.....	102
2.16	Overloading trong Java .....	104
2.17	Overriding trong Java .....	106
2.18	Lớp trừu tượng trong Java.....	107
2.19	Interface trong Java .....	108
2.20	Tìm hiểu về tính chất hướng đối tượng (Object Oriented Programming) trong Java.....	111
2.20.1	Tính Trừu tượng ( <i>abstraction</i> ) .....	112
2.20.2	Tính đóng gói ( <i>encapsulation</i> ).....	112
2.20.3	Tính kế thừa ( <i>inheritance</i> )trong Java.....	112
2.20.4	Tính đa hình ( <i>polymorphism</i> ).....	112
2.21	Chú ý trong truyền đối số vào method trong Java ( <b>Passing Data Type Arguments</b> ) .....	115
2.21.1	Truyền đối số <b>Primitive</b> .....	116
2.21.2	Truyền đối số <b>Reference</b> .....	116
2.22	Kiểu dữ liệu bao bọc ( <i>Wrapper classes</i> ).....	117

2.23	Autoboxing and Unboxing trong Java .....	121
2.24	Kiểu dữ liệu <b>String</b> .....	122
2.24.1	Replacing and Splitting Strings.....	125
2.24.2	Matching, Replacing and Splitting by Patterns.....	126
2.24.3	Conversion between Strings and Arrays.....	127
2.24.4	Formatting Strings.....	128
2.25	Kiểu dữ liệu <b>StringBuilder</b> và <b>StringBuffer</b> .....	129
2.26	Tạo, biên dịch và chạy ứng dụng Java đơn giản trên môi trường window mà không cần IDE .....	130
2.26.1	Cài đặt biến môi trường.....	130
2.26.2	Soạn và biên dịch, chạy mã nguồn Java.....	132
3	PHẦN 3 JAVA ADVANCE DÀNH CHO LẬP TRÌNH VIÊN .....	135
3.1	Kiểu dữ liệu <b>DateTime</b> và <b>formatter</b> .....	135
3.1.1	Nhận Date và Time hiện tại trong Java .....	136
3.1.2	Định dạng Date Time với <b>SimpleDateFormat</b> .....	136
3.2	Kiểu dữ liệu liệt kê <b>Enum Types</b> .....	139
3.3	Xử lý ngoại lệ (Exception Handling).....	141
3.4	Java IO/NIO.2.....	145
3.4.1	Hiểu biết về Java IO và NIO.2 .....	145
3.4.2	Làm việc với File trong Java (Handling Files in Java) .....	147
3.4.3	Làm việc với Stream trong Java.....	149
3.4.4	Readers and Writers.....	154
3.5	Collection .....	156
3.5.1	Collection Interface trong Java .....	156
3.5.2	List Interface và các lớp thực thi của List trong Java .....	159
3.5.3	Set Interface và các lớp thực thi của Set trong Java .....	162
3.5.4	Queue Interface trong Java.....	165
3.5.5	BlockingQueue Interface .....	167
3.6	Map trong Java .....	169
3.6.1	Map Interface và các lớp thực thi của Map .....	169
3.7	So sánh giữa các kiểu dữ liệu Implement của Collections.....	174
3.8	Lớp thư viện Collections trong Java .....	174
3.9	Lambda Expression .....	177
3.10	Stream API trong Java .....	179
3.10.1	Stream Song song (Parallel Stream) .....	181

3.10.2	Các cách tạo Stream:.....	182
3.11	Generic.....	183
3.12	Đa luồng (Multi Thread) trong Java .....	185
3.12.1	Tư duy lập trình đa luồng trong Java .....	185
3.12.2	Luồng (Thread) là gì.....	186
3.12.3	Creating a Thread Using the Thread Class .....	187
3.12.4	Creating a Thread Using the Runnable Interface .....	189
3.12.5	Vòng đời của Thread (Lifecycle of a Thread).....	190
3.12.6	Cơ chế Synchronization and Locks trong lập trình đa luồng.....	196
3.12.7	Cơ chế Wait và Notify trong Java .....	199
3.12.8	Cơ chế Wait và NotifyAll trong Java .....	203
3.12.9	Độ ưu tiên của Thread.....	207
3.12.10	Cơ chế đồng bộ Semaphore.....	207
3.12.11	Deadlock và các kỹ thuật tránh deadlock .....	210
3.12.12	missing signal.....	213
3.12.13	Thread pool trong Java .....	216
3.13	Lập trình mạng (Networking) trong Java:.....	222
3.13.1	Các khái niệm mạng (networking) cơ bản.....	222
3.13.2	Tư duy lập trình mạng (Networking) trong Java.....	225
3.13.3	Khởi tạo một Socket Server trong java .....	226
3.13.4	Khởi tạo một Client Socket trong java .....	227
3.13.5	Cơ chế 1 Server phục vụ nhiều Client.....	228
3.13.6	Truyền nhận dữ liệu Client-server socket.....	229
3.13.7	Truyền nhận một Object giữa client và server .....	229
3.13.8	<b>URL class trong java.....</b>	230
3.13.9	<b>URLConnection class trong java .....</b>	231
3.13.10	HttpURLConnection class trong java .....	231
3.13.11	InetAddress class trong java .....	233
3.13.12	DatagramSocket and DatagramPacket trong Java .....	234
3.14	Java và Database:.....	235
3.14.1	Tư duy lập trình giao tiếp Database .....	235
3.14.2	Cở sở dữ liệu quan hệ (RDB).....	235
3.14.3	Cở sở dữ liệu Không hệ (NoRDB).....	235
3.14.4	Truy vấn cơ sở dữ liệu quan hệ (SQL) .....	235

3.14.5 JDBC .....	235
- Sau đây chúng ta sẽ xét một ví dụ về việc truy cập Mysql Database sử dụng JDBC như sau:.....	236
set classpath=%classpath%;d:\lib\mysql-connector-java-8.0.13.jar; d:\lib\ojdbc7.jar .....	238
3.14.6 PreparedStatement.....	240
3.14.7 CallableStatement.....	241
3.15 API và Webservice trong Java.....	241
3.15.1 Hiểu biết về API.....	241
3.15.2 Hiểu biết về webservice .....	241
3.15.3 RESTful webservice .....	242
3.16 Bộ thu dọn rác (Garbage Collector) trong Java.....	249
3.17 Join dự án Maket Online. ....	252

## HƯỚNG DẪN SỬ DỤNG

Cuốn sách này là sự đúc kết từ kinh nghiệm tích lũy được từ quá trình sử dụng ngôn ngữ lập trình Java vào các dự án lớn nhỏ trong thực tế công việc, từ các giai đoạn huấn luyện các bạn thực tập sinh, thử việc và freshser và từ quá trình đào tạo cho các bạn sinh viên trong nhiều năm. Vì vậy đây không phải cuốn sách chứa đựng toàn bộ nội dung về Java mà là cuốn sách giúp các bạn tiếp cận với Java nhanh và dễ dàng hơn đồng thời tập trung vào những nội dung kiến thức chính có tính ứng dụng cao, nhữn gì đúc rút được từ kinh nghiệm và những lời khuyên có được từ thực tế. Để học tốt nhất bộ sách này thì nên kết hợp với cuốn “ACTIVE STUDY-THUC HANH JAVA CORE.pdf”, “ACTIVE STUDY-QUIZ OF JAVA CORE.pdf” được download trên trang “<http://activestudy.online/>” và bộ video “ACTIVE STUDY-JAVA CORE” của bộ sách được Active Study xuất bản cùng. Bộ video sẽ đọc tài liệu cùng bạn và giải thích thêm về các vấn đề cho các bạn dễ hiểu hơn. Bộ video cũng sẽ cùng các bạn giải quyết các bài tập ứng dụng đi kèm.

Trong quá trình đọc tài liệu các bạn nên ứng dụng làm bài Quiz và bài tập ngay, Các bài tập nhất quyết phải được chính các bạn hiểu và tự mình gõ code và đừng chỉ copy, page, cách này ko mấy hiệu quả và bạn có thể sẽ phải trả giá thêm về thời gian học Java chứ không thể nhanh hơn được. Phương pháp học code hiệu quả nhất là tự tay code và thay đổi một số chỗ trong chương trình

để thấy được tác dụng của từng dòng code. Trong quá trình học với mỗi chủ đề các bạn nên tìm ra câu trả lời cho 3 câu hỏi sau:

- 1: Cái này là Là cái gì ?
- 2: Cái này làm thế nào?
- 3: Cái này để làm gì?

Chúc các bạn có được những giờ học thú vị và hiệu quả với Java. Đừng quên thăm Fanpage và website của Active Study nhé:

Website: <http://activestudy.online>

Fanpage: <https://www.facebook.com/activestudy.edu.vn>

Support Group (Active Study member only):

<https://www.facebook.com/groups/ActiveStudy.JavaCoreBasic.Support>

Author : Hải Tân – Active Study

## JAVA CORE CHO MỌI NGƯỜI

### 1 PHẦN 1 KHÁI NIỆM LẬP TRÌNH CHO NGƯỜI MỚI BẮT ĐẦU

#### 1.1 Chương trình phần mềm là gì

- Chương trình phần mềm là một tập hợp các đoạn mã (code) đã được biên dịch và đóng gói thành một đơn vị gọi là chương trình, Chương trình sẽ được máy tính hiểu và chạy.
- Khi sử dụng máy tính là chúng ta chạy các phần mềm trên phần cứng máy tính
- Phần mềm lớn nhất và đặc biệt nhất mà chúng ta hay dùng đó là hệ điều hành window. Ngoài ra các phần mềm thông thường khác thì chúng ta hay chạy thông qua hệ điều hành của máy tính.

#### 1.2 Lập trình là gì

- Lập trình là quá trình chúng ta sử dụng một ngôn ngữ lập trình để viết và biên dịch thành một chương trình máy tính.

#### 1.3 Ngôn ngữ lập trình là gì

- Ngôn ngữ lập trình là một loại ngôn ngữ đặc biệt được thiết kế để giúp các lập trình viên có thể dựa trên đó viết các chỉ dẫn để máy tính thực hiện một hoặc nhiều tác vụ cho trước.
- Ngôn ngữ lập trình định nghĩa một bộ các quy tắc viết mã lệnh (còn gọi là cú pháp) để lập trình viên có thể dựa vào đó viết các chỉ dẫn thực hiện các tác vụ cụ thể cho máy tính. Các ngôn ngữ lập trình khác nhau có các quy tắc riêng khác nhau.
- Các loại ngôn ngữ lập trình phổ biến có thể kể đến như sau: pascal, C, C++, Java, JavaScript, Paython, kotlin, objective C, Swift, PHP, ASP.net, C#.net, VB.net.... rất rất nhiều. Chúng ta sẽ tìm hiểu về công dụng của một vài ngôn ngữ phổ biến ở các phần sau

#### 1.4 Các phương pháp lập trình

##### 1.4.1 Phương pháp lập trình tuần tự (Sequential Programming)

- Đây là phương pháp lập trình đầu tiên vào thủa sơ khai. Một chương trình sẽ là một tập hợp các câu lệnh có thứ tự chạy nối tiếp nhau.

- Chương trình sẽ chạy từ câu lệnh đầu tiên, và kết thúc ở câu lệnh cuối cùng. Lập trình thế này khó, lâu, và việc xử lý các bài toán lớn sẽ khó khăn, chương trình chỉ có thể viết theo một cấu trúc duy nhất là tuần tự.

#### 1.4.2 Phương pháp lập trình cấu trúc (Structured Programming)

- Đây là một phần của các phương pháp lập trình thủ tục, ở phương pháp lập trình này ngoài cấu trúc tuần tự, ta có thể viết chương trình theo các cấu trúc rẽ nhánh, cấu trúc vòng lặp và kết hợp của các cấu trúc đó với nhau. Ở phương pháp lập trình này xuất hiện khái niệm về cấu trúc điều khiển (control structure) như if-else, switch case... để khiển rẽ nhanh hay for, while để điều khiển vòng lặp.
- Phương pháp này đã khiến việc lập trình trở lên linh hoạt hơn rất nhiều, nhưng nó vẫn thực sự hạn chế khi lập trình các hệ thống lớn. Hiện nay nó không còn là phương pháp lập trình thông dụng. Nhưng nó vẫn tồn tại như là một phần không thể thiếu trong các phương pháp lập trình hiện đại hơn.

#### 1.4.3 Phương pháp lập trình thủ tục(Procedural Programming)

- Phương pháp lập trình thủ tục hay còn gọi là hướng chức năng. Phương pháp này sẽ chia một chương trình thành các khối thủ tục nhỏ, mỗi thủ tục sẽ có dữ liệu và logic riêng. Các thủ tục làm việc độc lập với nhau, dữ liệu sẽ được trao đổi qua các đối số (arguments) và giá trị trả về (returned value). Việc chia thủ tục sẽ có nhiều lợi thế ví dụ như việc sử dụng lại mã (reuse code) dễ dàng chia công việc cho nhiều người.
- Lập trình thủ tục đi liền với các khái niệm khái niệm hàm (function), thủ tục (procedure), tham số (paramenter), đối số (argument), trả về (return).. Ở một số ngôn ngữ như C, C++ không phân biệt hàm với thủ tục, ở một số ngôn ngữ khác như pascal thì hàm sẽ có giá trị trả về và sử dụng từ khóa function, còn thủ tục không có giá trị trả về và sử dụng từ khóa procedure.
- Cách lập trình này đã triều tượng hóa công việc lập trình hơn rất nhiều. Người lập trình có thể quan sát tổng thể một chương trình qua các chức năng của nó mà không cần quan tâm tới chi tiết trong nó. Hiểu được đặc điểm này là rất quan trọng, nó sẽ giúp bạn giải quyết các bài toán lớn một cách dễ dàng và chính xác hơn rất nhiều.
- Có thể lập trình thủ tục khó tiếp cận và không có được nhiều ưu điểm như phương pháp hướng đối tượng nhưng nó là phương pháp lập trình quan trọng. Nắm được phương pháp này bạn sẽ dễ

dàng tiếp cận được với những phương pháp lập trình cao hơn, lập trình hướng đối tượng dù là phương pháp hiện đại hơn nhưng nó vẫn chưa đựng trong đó các hàm và thủ tục.

- Các ngôn ngữ hỗ trợ lập trình hướng thủ tục thông dụng như: C, C++, Pascal, PHP...

#### 1.4.4 Phương pháp lập trình hướng đối tượng (Object-Oriented Programming)

- Lập trình hướng đối tượng (object-oriented programming viết tắt là OOP), hay còn gọi là lập trình định hướng đối tượng, là kĩ thuật lập trình xây dựng và sử dụng các đối tượng để giải quyết bài toán. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép lập trình viên tập trung vào các đối tượng phần mềm ở bậc cao hơn. Ngoài ra, nhiều người còn cho rằng OOP dễ tiếp thu hơn cho những người mới học về lập trình hơn là các phương pháp trước đó. Một cách giản lược, đây là khái niệm và là một nỗ lực nhằm giảm nhẹ các thao tác viết mã cho người lập trình, cho phép họ tạo ra các ứng dụng mà các yếu tố bên ngoài có thể tương tác với các chương trình đó giống như là tương tác với các đối tượng vật lý.
- Hầu hết các ngôn ngữ lập trình hiện đại đều có xu hướng hỗ trợ lập trình hướng đối tượng. Các ngôn ngữ lập trình hướng đối tượng phổ biến: Objective-C, C++, JAVA, C#...

#### 1.5 Lập trình ứng dụng máy tính (Destop APP)

- Các ứng dụng chạy trên máy tính thường được gọi là Ứng dụng desktop. Loại ứng dụng này chạy trực tiếp trên máy tính thông qua hệ điều hành. Một số ứng dụng như ứng dụng Microsoft Office desktop, Media player, teamview, zalo desktop.
- Để lập trình lên các ứng dụng desktop này thì lập trình viên thường sử dụng các ngôn ngữ lập trình như: C#.net, VB.net, C++, Java...
- Ngày nay với sự phát triển của hạ tầng mạng và công nghệ điện toán đám mây (Cloud computing) các ứng dụng desktop ngày càng được ảo hóa và chạy trên môi trường web base, đây cũng là xu thế của tương lai.

#### 1.6 Lập trình ứng dụng di động

- Các phần mềm chạy trên các thiết bị di động như điện thoại, tablet, IPad... thì gọi là các ứng dụng di động.

- Các ứng dụng này có đặc điểm là chạy trên các hệ điều hành dành riêng cho thiết bị di động như Android, IOS. Và việc tương tác với phần mềm của người dùng cũng như chạy trên các thiết kế phần cứng khác biệt so với máy tính đó là các thiết bị di động.
- Các ứng dụng di động phổ biến: Mobile game, mobile app nghe nhạc, xem file, Zalo mobile...
- Các ngôn ngữ lập trình thường được dùng để lập trình ứng dụng di động đó là Java, Kotlin, ObjectiveC, Swift, javascript...

### 1.7 Lập trình web

- Lập trình web là lập trình các ứng dụng giao tiếp với người dùng thông qua các webbrowser. Đó là chính là các website mà chúng ta hàng ngày vẫn dùng, vẫn tương tác.
- Ứng dụng web hết sức đặc biệt, đó là các ứng dụng được chạy trên một server được gọi là Hostting, hay còn gọi là web server. Người dùng sử dụng máy tính hoặc các thiết bị di động để truy cập vào các trang web chạy trên các máy chủ web này thông qua webbroser. Máy chủ trả về các dữ liệu theo định dạng HTML hoặc những đoạn code javaScript để webbrowser hiển thị cho người dùng tương tác.
- Ngày nay các ứng dụng ngày càng được chuyển sang ứng dụng web để chạy trên môi trường webbase vì tính tiện lợi của nó.
- Các ứng dụng web nổi tiếng có thể kể đến đó là trang google.com, facebook.com...
- Các ngôn ngữ lập trình phổ biến được sử dụng để xây dựng ứng dụng web đó là: Java, JavaScript, PHP, ASP.net, Paython...

### 1.8 Lập trình core service

- Là các ứng dụng chạy trên các máy chủ để thực hiện các chức năng dịch vụ ngầm hoặc để đáp ứng cho các ứng dụng khác gọi tới. Ứng dụng loại này không cần đòi hỏi phải có giao diện tương tác với người dùng cuối mà chủ yếu là các giao diện đơn giản cho người vận hành. Ứng dụng loại này thường yêu cầu tính bền bỉ, an toàn, bảo mật và hiệu năng cao để đáp ứng lượng lớn các giao dịch nghiệp vụ, vì vậy đây là một lĩnh vực lập trình khó đòi hỏi kỹ năng và tư duy lập trình tốt.
- Các ứng dụng có thể kể tới như: Ứng dụng core banking, core chứng khoán, telecom, blockchain...
- Các loại ngôn ngữ lập trình phổ biến được sử dụng để lập trình lên loại ứng dụng này đó là: Java, C++, paython...

## 1.9 Lập trình nhúng

- Đây là các ứng dụng khá đặc biệt vì các chương trình dạng này sẽ được nạp vào các con IC (chíp) và chạy trên các thiết bị điện tử. Môi trường rất khiêm tốn về tốc độ và bộ nhớ cũng như các chuẩn giao tiếp rất đặc biệt. Việc lập trình nhúng đòi hỏi phải có kỹ năng quản lý bộ nhớ tốt, chương trình viết ra khi chạy sử dụng bộ nhớ nhỏ nhất có thể.
- Các ứng dụng có thể kể đến đó là chương trình điều khiển máy giặt, máy lau nhà, smart home, chương trình điều khiển chạy trên ô tô...
- Các ngôn ngữ phổ biến được dùng để làm nhúng đó là: C, Java...

## 1.10 Công cụ soạn thảo code(IDE) phổ biến

- Netbean: Dùng để code các ứng dụng Java, JavaWeb, C, C++, PHP, HTML, JavaScript
- Eclip: Dùng để code các ứng dụng Java, JavaWeb, C, C++, PHP, HTML, JavaScript
- IntelliJ: Dùng để code các ứng dụng Java, JavaWeb, C, C++, HTML, Kotlin, Groovy, Java Android
- Visual studio: ASP.net, C#.net, VB.net, J#.net, Visual C++
- Androi studio: Java Android, Kotlin
- Xcode: Objective C, Swift.

## 1.11 Các thuật ngữ sử dụng trong lập trình

### 1.11.1 Ngôn ngữ lập trình bậc thấp

- Trong khoa học máy tính, ngôn ngữ lập trình bậc thấp là một ngôn ngữ lập trình liên quan chặt chẽ đến phần cứng máy tính. Từ "thấp" không có nghĩa là ngôn ngữ này kém hơn các ngôn ngữ lập trình bậc cao mà điều này nghĩa là các lệnh của nó rất gần ngôn ngữ máy, chính vì thế nó không thân thiện(dễ hiểu) đối với lập trình viên.
- Các từ "bậc cao" và "bậc thấp" còn sử dụng với ý nghĩa tương đối; một lập trình viên Java có thể xem ngôn ngữ C là ngôn ngữ lập trình bậc thấp.
- Các ngôn ngữ lập trình bậc thấp thường được chia thành hai loại: thế hệ thứ nhất và thế hệ thứ hai.
  - o Ngôn ngữ lập trình thế hệ thứ nhất, hay 1GL, là mã máy. Nó là ngôn ngữ duy nhất mà bộ vi xử lý có thể hiểu. Hiện nay các lập trình viên hầu như không bao giờ viết chương trình trực

tiếp bằng ngôn ngữ máy vì nó không chỉ yêu cầu chú ý nhiều đến các chi tiết mà một ngôn ngữ bậc cao xử lý một cách tự động mà còn yêu cầu ghi nhớ và tìm những mã lệnh bằng số cho mỗi chỉ thị được sử dụng.

- Ngôn ngữ lập trình thế hệ thứ hai, hay 2GL, là ngôn ngữ assembly. Nó được xem là ngôn ngữ thế hệ thứ hai vì mặc dù nó không phải là ngôn ngữ máy nhưng lập trình viên vẫn phải hiểu về kiến trúc của bộ vi xử lý (như các thanh ghi và các lệnh của bộ vi xử lý). Những câu lệnh đơn giản được dịch trực tiếp ra mã máy.

#### 1.11.2 Ngôn ngữ lập trình bậc cao

- Trong khoa học máy tính, một ngôn ngữ lập trình bậc cao (high-level programming language) là một ngôn ngữ lập trình có sự trừu tượng hóa mạnh mẽ khỏi các chi tiết của máy tính. So với các ngôn ngữ lập trình bậc thấp (low-level programming language), nó có thể sử dụng các yếu tố ngôn ngữ tự nhiên, dễ sử dụng hơn, hoặc có thể tự động (hoặc thậm chí che giấu hoàn toàn) các khu vực quan trọng của các hệ thống điện toán (ví dụ, quản lý bộ nhớ (memory management)), làm quá trình phát triển chương trình đơn giản hơn và tương đối dễ hiểu hơn so với một ngôn ngữ bậc thấp.
- Ngôn ngữ bậc cao đầu tiên được xây dựng vào năm 1956 là ngôn ngữ FORTRAN. Ngày nay có rất nhiều các ngôn ngữ lập trình bậc cao như PASCAL, C, C++, ASP.net, C#.net, VB.net, Foxpro, PHP, javaScript, Paython... và đặc biệt là Java.

#### 1.11.3 Mã nguồn

- Mã nguồn (source code) được hiểu trong tin học là một dãy các câu lệnh được viết bằng một ngôn ngữ lập trình (Java, C, C++...). Mã nguồn có thể được biên dịch ra mã máy hoặc mã trung gian.

#### 1.11.4 Mã máy

- Mã máy hay còn được gọi là Ngôn ngữ máy (machine language hay machine code) là một tập các chỉ thị dạng nhị phân được CPU của máy tính trực tiếp thực thi. Mỗi chỉ thị thực hiện một chức năng xác định, ví dụ như tải dữ liệu, nhảy hay tính toán số nguyên trên một đơn vị dữ liệu của thanh ghi CPU hay bộ nhớ. Tất cả các chương trình được thực thi trực tiếp bởi CPU đều là các chuỗi các chỉ thị này.

- Mã máy nhị phân (khác với mã hợp ngữ) có thể được xem như là phương thức biểu diễn thấp nhất của một chương trình đã biên dịch hay hợp dịch, hay là ngôn ngữ lập trình nguyên thủy phụ thuộc vào phần cứng (ngôn ngữ lập trình thế hệ đầu tiên).
- Hiện nay, hầu như tất cả các chương trình máy tính trong thực tế đều được viết bằng các ngôn ngữ bậc cao hay (đôi khi) hợp ngữ, và sau đó được dịch thành mã máy thực thi bằng các công cụ hỗ trợ như trình biên dịch, trình hợp dịch hay trình liên kết. Ngoài ra, các chương trình được viết bằng ngôn ngữ thông dịch thì được dịch sang mã máy nhờ trình thông dịch tương ứng.

#### 1.11.5 Trình Biên dịch chương trình phần mềm

- Trình biên dịch, còn gọi là phần mềm biên dịch (compiler), là một chương trình máy tính làm công việc dịch một chuỗi các câu lệnh được viết bằng một ngôn ngữ lập trình (gọi là ngôn ngữ nguồn hay mã nguồn), thành một chương trình tương đương nhưng ở dưới dạng một ngôn ngữ máy tính mới (gọi là ngôn ngữ đích) và thường là ngôn ngữ ở cấp thấp hơn như mã trung gian hoặc mã máy.
- Mỗi một loại ngôn ngữ lập trình thì đều có một trình biên dịch riêng biệt của riêng nó. Và trình biên dịch thường đi kèm với bộ cài của bộ phát triển phần mềm.

#### 1.11.6 Trình Thông dịch chương trình phần mềm

- Trình thông dịch thực hiện biên dịch một chương trình nguồn theo từng phân đoạn. Sau đó, thực thi các đoạn mã đã được biên dịch. Hoàn toàn khác với trình biên dịch (Trình biên dịch thực hiện biên dịch hoàn toàn rồi mới thực thi chương trình).
- Khi phát triển phần mềm thì cần có trình thông dịch để thực hiện debug chương trình.

#### 1.11.7 Bug và Debug (lỗi phần mềm và Gỡ lỗi)

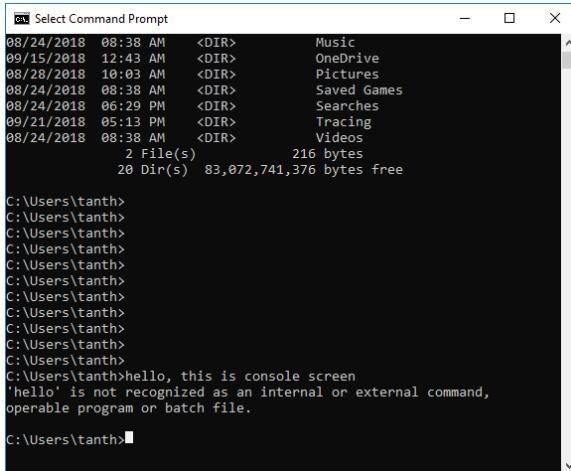
- Lỗi phần mềm trong chương trình làm cho kết quả không chính xác hoặc không mong muốn được gọi là "bug". Các bug thường là các lỗi logic. Quá trình sửa lỗi được gọi là "debug" và thường sử dụng kỹ thuật hoặc công cụ chính thức để xác định lỗi (bug)

### 1.11.8 Exception (Ngoại lệ)

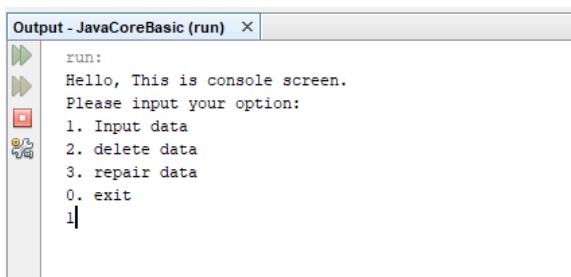
- Là các lỗi phần mềm xảy ra trong quá trình chạy. Các lỗi này được chương trình phát hiện và ném ra thành một Exception (ngoại lệ). Trong lập trình lập trình viên có những phương pháp để bắt và xử lý các ngoại lệ này để chương trình không bị dừng và vẫn đảm bảo tính logic của chương trình.

## 1.11.9 Console

- Là giao diện tương tác thông qua dòng lệnh text. Thông qua giao diện console ứng dụng nhận các lệnh hoặc dữ liệu đầu vào từ người dùng thông qua text và ứng dụng cũng xuất ra màn hình console các text, đôi khi đó là thông báo đôi khi nó là các log chương trình phần mềm.



- Hoặc đó chính là màn hình console trong lúc chúng ta chạy chương trình ứng dụng dạng console của các trình soạn thảo code



### 1.11.10 GUI (Graphical User Interface)

- Giao diện đồ họa người dùng trong tiếng Anh gọi tắt là GUI (Graphical User Interface) là một thuật ngữ trong ngành Công nghệ thông tin. Đó là một cách giao tiếp với máy tính hay các thiết bị điện tử bằng hình ảnh và chữ viết thay vì chỉ là các dòng lệnh đơn thuần. GUI được sử dụng phổ biến

trong máy tính, các thiết bị cầm tay, các thiết bị đa phương tiện, hoặc các linh kiện điện tử trong văn phòng, nhà ở...

- GUI được phát triển trong thập niên 1970. Ngày nay hầu hết các hệ điều hành máy tính nhiều người dùng đều sử dụng giao diện này.
- Trong phát triển ứng dụng phần mềm có tương tác với người dùng đầu cuối thì GUI rất được coi trọng về mặt hình thức và độ tiện dụng trong tương tác. Những lập trình viên chuyên phát triển phần GUI thì được gọi là lập trình viên Frontend.

#### 1.11.11 Database

- Database là một hệ quản trị cơ sở dữ liệu. Tại đây các dữ liệu được lưu trữ và truy cập và quản lý. Các thành phần chính của cơ sở dữ liệu bao gồm các bản ghi (record) dữ liệu gồm các trường dữ liệu được tập hợp lại và phân thành các đơn vị lớn hơn gọi là bảng(table) hoặc collection. Các table có thể có quan hệ ràng buộc với nhau chặc chẽ thì gọi là cơ sở dữ liệu quan hệ(SQL). Các collection không có sự ràng buộc về dữ liệu với nhau thì gọi là cơ sở dữ liệu không quan hệ hay còn gọi là noSQL.
- Để truy cập cơ sở dữ liệu thì cần mở kết nối và sử dụng các câu lệnh truy vấn, hoặc thêm, sửa, xóa các dữ liệu mong muốn.
- Trong Database ngoài các table (collections) thì còn nhiều những đối tượng khác phục vụ lưu trữ, sắp xếp, vận hành và truy vấn dữ liệu.
- Các loại cơ sở dữ liệu phổ biến như sau:
  - o Cơ sở dữ liệu quan hệ : Oracle SLQ, MySQL, Microsoft SQL, Microsoft Access, Foxpro
  - o Cơ sở dữ liệu không quan hệ: MongoDB, Oracle NoSQL, OrientDB, Redis

#### 1.11.12 Var (Biến)

- Biến là một thành phần quan trọng trong lập trình. Biến được khai báo trong chương trình phần mềm và được dùng để chứa dữ liệu tạm thời.
- Chương trình = giải thuật + dữ liệu. Dữ liệu chính là những giá trị được chứa đựng ở trong các biến.
- Chúng ta gọi là biến là bởi vì biến có tên và có thể thay đổi được giá trị mà nó chứa đựng.

### 1.11.13 Funtion, Procedure, Method (Hàm, Thủ tục, Phương thức)

- Trong lập trình 3 khái niệm Funtion, Procedure, Method gần tương tự nhau. Chúng đều là một đơn vị tổ chức mã nguồn, các mã code được chia nhỏ thành các đơn vị nhỏ để thực hiện một giải thuật nhất định và được đại diện bởi một cái tên. Trong đó quy định rõ dữ liệu đầu vào và đầu ra, khi triệu gọi tới Funtion, Procedure hoặc method thì chúng ta dùng tên đại diện và truyền các dữ liệu đầu vào đồng thời nhận lại dữ liệu đầu ra sau khi thực hiện xong đoạn mã code này.
- Sự khác nhau của Funtion, procedure và method như sau:
  - o Funtion có đầu vào và trả ra
  - o Procedure: có đầu vào mà không có dữ liệu trả ra
  - o Method: Ngày nay trong hầu hết các ngôn ngữ lập trình hiện đại không còn phân biệt function và procedure nữa mà gọi chung chúng là method. Vậy sau đây thì chúng ta gọi chung chúng là method hoặc phương thức.
- Xét ví dụ sau để thấy một method trong Java. Tên của method là `login()`, Dữ liệu đầu ra là kết quả login có kiểu `boolean` Dữ liệu đầu vào là `user` và `password`.

```

  public boolean login(String user, String password) {
50      for (int i = 0; i < MAX_MEMBER; i++) {
51          if ((members[i] != null) && (members[i].login(user, password) == true)) {
52              currentPerson = members[i];
53              break;
54          }
55      }
56      if (currentPerson != null) {
57          System.out.println("Login Success.");
58          return true;
59      } else {
60          System.out.println("Login false.");
61      }
62  }
63

```

### 1.11.14 Giải thuật

- Giải thuật (hay còn gọi là thuật toán - tiếng Anh là Algorithms) là một tập hợp hữu hạn các chỉ thị để được thực thi theo một thứ tự nào đó để thu được kết quả mong muốn. Nói chung thì giải thuật là độc lập với các ngôn ngữ lập trình, tức là một giải thuật có thể được triển khai trong nhiều ngôn ngữ lập trình khác nhau.
- Xuất phát từ quan điểm của cấu trúc dữ liệu, dưới đây là một số giải thuật quan trọng:
  - o Giải thuật Tìm kiếm: giải thuật tìm kiếm dữ liệu.
    - Tìm kiếm tuần tự

- Tìm kiếm nội suy
- Tìm kiếm nhị phân
- Giải thuật Sắp xếp: Giải thuật để sắp xếp các phần tử theo thứ tự nào đó như tăng dần hoặc giảm dần...
  - Bubble sort.
  - Quick sort.
  - Simple selection sort.
  - Heap sort.
  - Simple insertion sort.
  - Shell sort.
  - Merge sort.
  - Radix sort.
- Giải thuật có thể được thi triển ở các ngôn ngữ lập trình khác nhau trong các methods.

#### 1.11.15 Argument (Đối số)

- Nói tới đối số là nói tới việc gọi method.
  - Đối số là các biến truyền vào các method và chúng được gọi là dữ liệu đầu vào của các method.
- Xét một ví dụ trong Java thì 2 biến `user` và `password` tại dòng 35 được gọi là đối số ([Argument](#)) của method `login`.

```

28 |         String user;
29 |         String passwrod;
30 |         Scanner input = new Scanner(System.in);
31 |         System.out.println("User\t: ");
32 |         user = input.nextLine();
33 |         System.out.println("Password\t: ");
34 |         passwrod = input.nextLine();
35 |         return login(user, passwrod);
  
```

#### 1.11.16 Parameter (Tham số)

- Nói tới tham số (Parameter) là nói tới bên trong của method.
- Tham số là các biến số chứa đựng các dữ liệu đầu vào được truyền cho method, và tham số được sử dụng nội bộ trong các method.
- Xét ví dụ sau để thấy rõ hơn.

```

public boolean login(String user, String password) {
    50   for (int i = 0; i < MAX_MEMBER; i++) {
    51       if ((members[i] != null) && (members[i].login(user, password) == true)) {
    52           currentPerson = members[i];
    53           break;
    54       }
    55   }
    56   if (currentPerson != null) {
    57       System.out.println("Login Success.");
    58       return true;
    59   } else {
    60       System.out.println("Login false.");
    61   }
    62   return false;
    63 }

```

Từ ví dụ trên ta thấy user và password là 2 parameter của method `boolean login(String user, String password)`

#### 1.11.17 Test (Kiểm thử phần mềm)

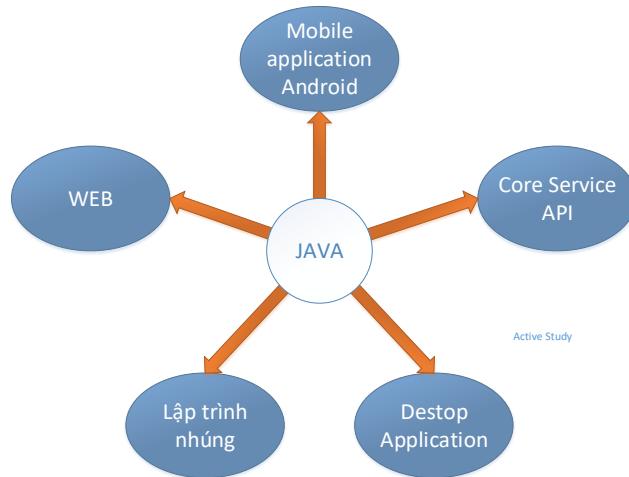
- Người ta thường nói phần mềm thì không bao giờ hết lỗi (bug).
- Trong quá trình phát triển phần mềm thì sẽ tồn tại ít hoặc nhiều những lỗi phát sinh do nhiều nguyên nhân chủ quan hoặc khách quan. Nguyên nhân chủ quan như là các lỗi về logic giải thuật, lỗi chức năng. Các lỗi khách quan như: lỗi thao tác người dùng, lỗi môi trường chạy và xung đột hoặc thiếu tài nguyên hệ thống...
- Kiểm thử phần mềm là quá trình dùng các phương pháp cụ thể để tìm kiếm, phát hiện ra các lỗi để giúp lập trình viên debug và vá lỗi.
- Các phương pháp test thì có rất nhiều thì khái quát có 3 loại như sau:
  - o Black box: Coi chương trình như một hộp đen và sử dụng chức năng của chúng để tìm ra lỗi
  - o White box: Mổ xẻ chương trình để dự đoán và tìm kiếm lỗi, Phương pháp này đòi hỏi tester cần phải có những kỹ năng về đọc code, truy vấn database, đọc và hiểu về luồng xử lý logic chương trình...
  - o Grey box: phương pháp kết hợp blackbox và white box.

## 2 PHẦN 2 JAVA BASIC DÀNH CHO NGƯỜI MỚI BẮT ĐẦU

### 2.1 Tại sao chúng ta lại nên lập trình bằng Java

- Java là ngôn ngữ lập trình có tính ứng dụng cao, được sử dụng vào hầu hết các lĩnh vực như: Lập trình web, Lập trình ứng dụng di động Android, Lập trình ứng dụng core service, Ứng dụng desktop

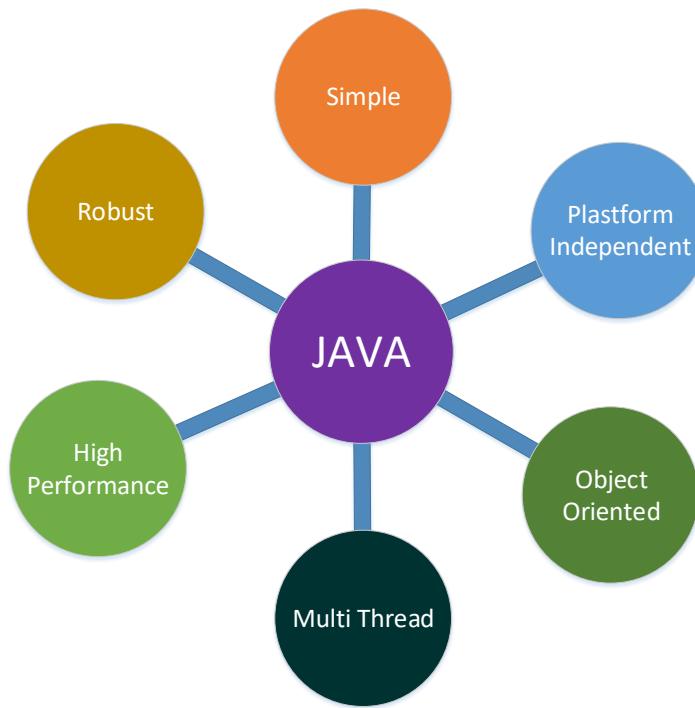
và lập trình nhúng. Vì vậy Java đang là ngôn ngữ được sử dụng nhiều vào bậc nhất trong những năm gần đây và lập trình viên Java cũng được các công ty phần mềm tìm kiếm rất nhiều. Cứ có 10 Job thì có tới 6 Job về Java rồi, mà mức thu nhập cũng rất hấp dẫn từ vài trăm tới vài ngàn Đô. Vì vậy chúng ta cứ yên tâm học lập trình Java đi



- Java chạy đa nền tảng vì vậy chúng ta sẽ tiết kiệm được thời gian và công sức phát triển sản phẩm và được các nhà phát triển phần mềm ưu tiên sử dụng trong sản xuất phần mềm.
- Java là ngôn ngữ lập trình bậc cao, gần gũi với ngôn ngữ con người vì vậy nó dễ tiếp cận đối với người mới bắt đầu và dần được lựa chọn làm ngôn ngữ lập trình nhập môn cho sinh viên các trường đại học trên thế giới.
- Đó là 3 lý do chính mà chúng ta nghiên cứu về Java.

## 2.2 Tìm hiểu đặc điểm ngôn ngữ lập trình java trước khi lập trình Java

- Trước lúc lập trình Java chúng ta nên tiềm hiểu về đặc điểm của ngôn ngữ này, nó sẽ giúp cho quá trình cài đặt và học ngôn ngữ này của chúng ta được nhanh hơn và trong các buổi phỏng vấn ai đó sẽ hỏi chúng ta những câu đại loại như: "Theo em thì ngôn ngữ Java có những đặc điểm gì nổi bật". Java có nhiều đặc điểm nhưng chúng ta chú ý đến 6 ưu điểm chính như hình sau:



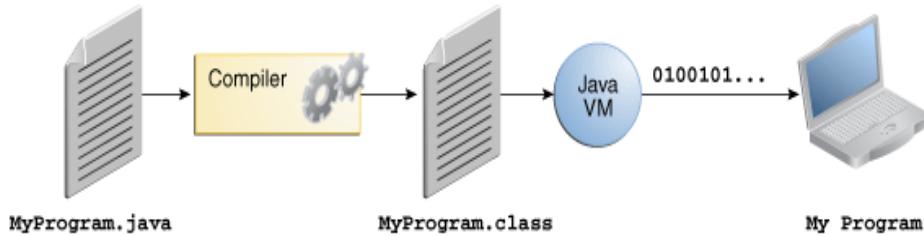
### 2.2.1 Ngôn ngữ Java đơn giản dễ dùng

- Java là ngôn ngữ có cú pháp theo dòng C, Kế thừa chọn lọc và cải tiến theo hướng đơn giản dễ dùng
- loại bỏ các yếu tố phức tạp đó là khái niệm con trỏ.
- Loại bỏ cấu trúc “struct” và “union” của C và chỉ cho phép đa kế thừa đối với Interface, và đơn kế thừa đối với class.
- Đặc biệt với bộ dọn rác “Garbage collection” giúp cho lập trình viên không phải đi thu hồi bộ nhớ trong quá trình lập trình, vì vậy ứng dụng sẽ tránh được rất nhiều các lỗi về Memory làm cho chương trình trở nên an toàn và quá trình phát triển sẽ tiết kiệm thời gian hơn rất nhiều.

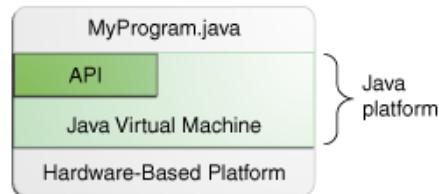
### 2.2.2 Ngôn ngữ Java Độc lập nền tảng

- Một trong những thế mạnh lớn nhất của Java là độc lập nền tảng cả phần cứng và phần mềm. Lập trình viên viết code một lần và dịch ra Bytecode sau đó có thể chạy trên mọi nền tảng hệ điều hành từ Window, Linux, Sun, Solaris, Mac/OS ...mọi loại phần cứng 32 hay 64 bit của các hãng khác nhau, Sở dĩ có được điều tuyệt vời này là nhờ vào các máy ảo JVM có ở khắp các nền tảng. Tính chất này đối với lập trình viên thì được cô đọng trong một câu nói đó là: “Write Once and

Run Anywhere". Điều này giảm thiểu được rất nhiều công sức của các nhà phát triển phần mềm vì không phải đi viết lại chương trình trên từng nền tảng, điều này cũng tạo lên tính đồng nhất, tính linh động của sản phẩm phần mềm viết bằng Java.



- Trong quá trình lập trình, Lập trình viên sử dụng các Java API đồng nhất và khi chạy thì được JVM xử lý theo các hướng khác nhau để cho ra cùng một kết quả giống nhau trên các nền tảng khác nhau nhau.



### 2.2.3 Java hướng đối tượng hoàn toàn

- Java hoàn toàn hướng đối tượng, phương pháp lập trình này tạo cho Java sức mạnh và lợi thế to lớn trong việc lựa chọn ngôn ngữ để giải quyết các bài toán mang tính chất phức tạp, rộng lớn và gần gũi với tư duy đời sống thực tế. Tính hướng đối tượng làm cho chương trình viết bằng Java trở nên gọn gàng, trong sáng, dễ hiểu dễ bảo trì, giảm thiểu thời gian phát triển và tránh được lượng lớn các lỗi logic trong lập trình.
- Chúng ta sẽ dành riêng một buổi để nghiên cứu về tính chất hướng đối tượng của Java. Để hiểu về tính chất hướng đối tượng của Java thì không khó, cái khó là áp dụng được các tính chất này vào các bài toán thực tế. Sau này khi nghiên cứu thêm về Java Design Pattern thì chúng ta sẽ thấy rất nhiều những vận dụng của hướng đối tượng vào các Design Pattern.

#### 2.2.4 Lập trình xử lý đa luồng (Multi-thread)

- Giống như C++ và một số ngôn ngữ lập trình cấp cao khác, Java cung cấp các lớp đối tượng cho phép khai báo nhiều tiểu tiến trình (Thread) trong chương trình để cùng xử lý song song một hoặc nhiều việc cùng một lúc. Điều này giúp cho chương trình viết bằng Java tăng được hiệu năng sử dụng CPU giúp tăng tốc độ chương trình

#### 2.2.5 Hiệu năng cao (High performance)

- Khác với các ngôn ngữ bậc thấp hoặc native khác chạy trực tiếp trên hệ điều hành (OS) thì chương trình viết bằng Java phải chạy thông qua JVM, điều này làm rắc rối hơn khi chạy một chương trình Java và làm giảm thiểu tốc độ của chương trình, thế nhưng theo như sự so sánh về tốc độ thì ở các phiên bản sau Java càng được cải thiện đáng kể về tốc độ xử lý đặc biệt từ bản JDK 1.6 trở đi, đặc biệt về tốc độ xử lý chuỗi nhanh hơn hẳn các ngôn ngữ khác như C++

#### 2.2.6 Mạnh mẽ (Robust)

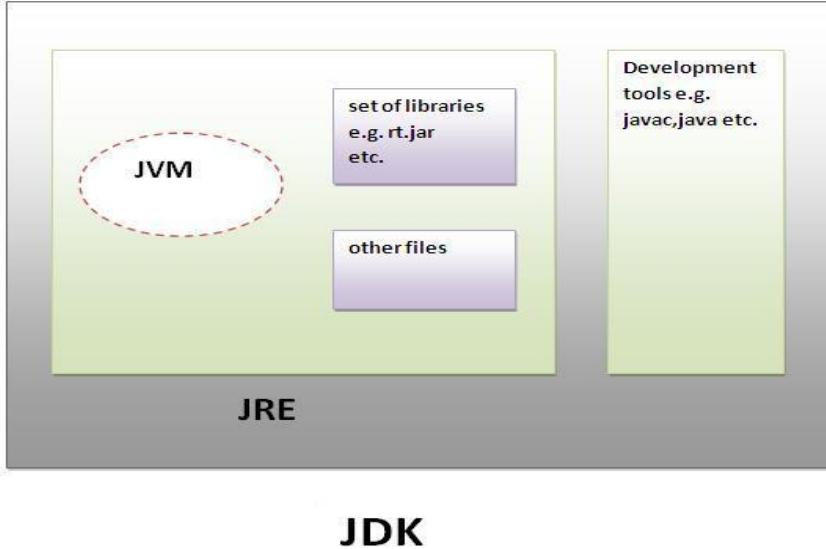
- Java cung cấp cho lập trình viên bộ thư viện API đầy đủ và tiện ích như Garbage Collection, Exception Handling... đồng thời hệ thống các lib mở rộng từ cộng đồng phát triển Java cũng rất khổng lồ tạo cho Java trở thành một gã khổng lồ mạnh mẽ.

### 2.3 Tìm hiểu về Kiến trúc Java

Java có 3 thành phần bao gồm:

- **Java Virtual Machine (JVM):** Máy ảo Java là linh hồn của Java, JVM giúp thông dịch và chạy các chương trình Java từ Bytecode. Với các nền tảng khác nhau sẽ có các máy ảo Java khác nhau trên nền tảng đó. Vì vậy khi cài đặt môi trường chạy Java thì cần chú ý chọn đúng bộ JRE hoặc JDK của nền tảng đó ví dụ như Window, Linux, Solaris, Mac... nền tảng 32 hay 64 bit.
- **Java SE Runtime Environment (JRE):** Là bộ phần mềm cung cấp môi trường để chạy các ứng dụng viết bằng Java, nó bao gồm cả JVM và tập hợp các thư viện cùng các file mà JVM sử dụng trong quá trình chạy vì vậy muốn triển khai một chương trình viết bằng Java thì cần phải cài đặt bộ JRE.

- **Java Development Kit (JDK)** : Là bộ công cụ cung cấp cho lập trình viên biên dịch và chạy chương trình Java. Trong bộ JDK bao gồm JRE và trình biên dịch javac, JavaDoc, Java Debugger, ... Vì vậy nếu muốn lập trình java thì chúng ta cần phải cài đặt bộ JDK tương ứng trên máy.



## 2.4 Cài đặt môi trường để lập trình Java

- Để bắt tay vào học lập trình Java thì trên môi trường nào cũng phải tiến hành cài đặt môi trường phát triển JDK và chọn một bộ soạn thảo code để quản lý và viết code cho project, bộ soạn thảo này được gọi là IDE (Integrated Development Environment), vậy chúng ta sẽ đi vào thực hiện ngay hai bước này để có thể bắt đầu viết một chương trình Java đầu tay.

### 2.4.1 Cài đặt JDK lên máy phát triển

- ❖ Download bộ JDK trên môi trường tương ứng tại Link chính thức sau của Oracle:

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

Sau khi cài đặt thì bộ JDK đã được lưu trong một thư mục dạng như sau: “C:\Program Files\Java\jdk1.8.0\_131”. Thư mục này ta sẽ gọi là JAVA\_HOME.

Name	Date modified	Type	Size
bin	7/16/2017 2:10 PM	File folder	
db	7/16/2017 2:10 PM	File folder	
include	7/16/2017 2:10 PM	File folder	
jre	7/16/2017 2:10 PM	File folder	
lib	7/16/2017 2:10 PM	File folder	
COPYRIGHT	3/15/2017 3:44 PM	File	4 KB
javafx-src.zip	7/16/2017 2:10 PM	WinRAR ZIP archive	4,978 KB
LICENSE	7/16/2017 2:10 PM	File	1 KB
README.html	7/16/2017 2:10 PM	Firefox HTML Doc...	1 KB
release	7/16/2017 2:10 PM	File	1 KB
src.zip	3/15/2017 3:44 PM	WinRAR ZIP archive	20,761 KB
THIRDPARTYLICENSEREADME.txt	7/16/2017 2:10 PM	Text Document	173 KB
THIRDPARTYLICENSEREADME-JAVAFX.txt	7/16/2017 2:10 PM	Text Document	63 KB

- Như vậy là chúng ta đã hoàn thành cài đặt bộ JDK và biến môi trường để sẵn sàng sử dụng, tiếp theo chúng ta sẽ lựa chọn cho mình một IDE phù hợp nhất nhé.

#### 2.4.2 Tìm kiếm một bộ soạn thảo code phù hợp nhất

- Về nguyên tắc chúng ta có thể sử dụng mọi trình soạn thảo như: Notepad, Notepad++... để soạn thảo mã Java sau đó trực tiếp gọi trình biên dịch Javac để biên dịch mã nguồn. Như vậy để chúng ta hiểu về bản chất của việc lập trình và không bị phụ thuộc hoàn toàn vào các IDE chuyên nghiệp như: Eclip, Netbean, IntelliJ IDE... Các IDE chuyên nghiệp này sẽ giúp chúng ta được các công việc sau để giảm thiểu thời gian phát triển:
  - Tạo Project để quản lý code
  - Liên kết với trình biên dịch Javac để biên dịch và thông báo kết quả biên dịch thay vì lập trình viên phải tự đi gọi Javac để biên dịch sau đó tự đọc kết quả sau biên dịch trong một cửa sổ dạng console, việc này làm cho việc biên dịch trở lên nhanh và tiện hơn rất nhiều.
  - Tự động dò tìm và cảnh báo các lỗi cú pháp và một số lỗi logic trước khi gọi trình biên dịch giúp cho lập trình viên được cảnh báo sớm và sửa chữa.
  - Recomment code và Auto complele code giúp lập trình viên soạn thảo code nhanh hơn, điều này rất là tuyệt vời thể hiện sự thông minh của một trình soạn thảo code.
- Vì vậy mà tất cả các lập trình viên chuyên nghiệp đều sử dụng một trong các IDE chuyên nghiệp này, Các bạn có thể lựa chọn một trong các IDE theo gợi ý ở trên, cái nào cũng được, chúng đều được thiết kế có các tính năng tương tự nhau, tiện ích gần tương tự nhau và hỗ trợ đầy đủ đối cho việc lập trình Java. Sau đây chúng ta sẽ tìm hiểu đôi chút về đặc điểm các IDE để lựa chọn cho phù hợp:

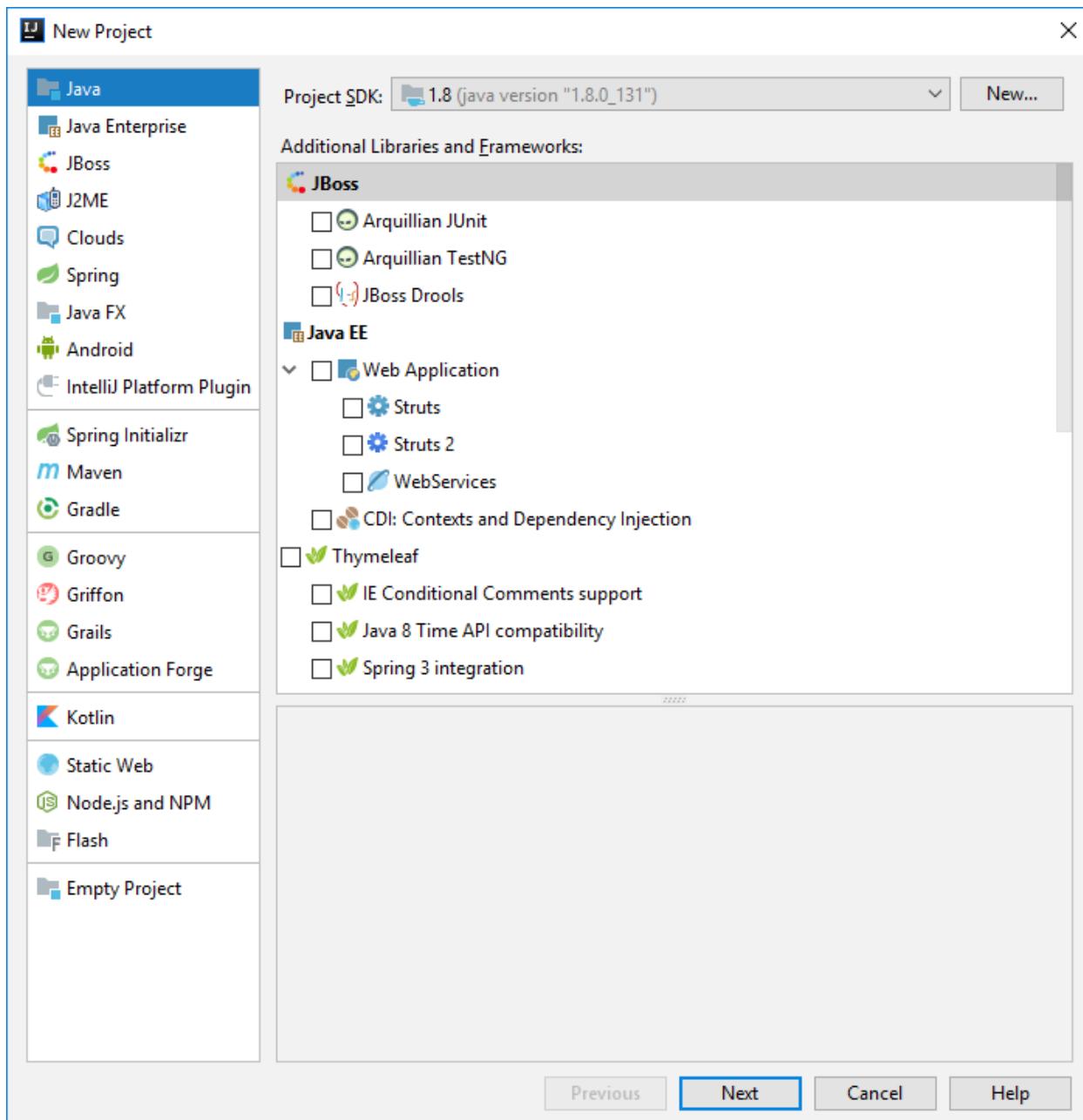
❖ **IntelliJ IDE:** IntelliJ là một IDE tương đối thông minh và hỗ trợ nhiều ngôn ngữ hơn và đang phát triển. JetBrains cũng là cha đẻ của Kotlin một ngôn ngữ phát triển ứng dụng di động Android mới được ra mắt lần đầu năm 2016. Các loại dự án trình IDE này hỗ trợ như sau: Java (Java SE, JavaEE, JavaME, JavaFX), Kotlin, NodeJS, Groovy, Scala và đối với bản Commercial Cũng hỗ trợ đầy đủ các framework và thư viện như Spring, Hibernate, struts, vaadin, webservice Restful và SOAP, websocket, JSON... nói chung là rất nhiều. Ngoài ra còn tích hợp rất tốt với các trình quản lý code như Git, SVN, Maven, Gradle...

➤ Link download: <https://www.jetbrains.com/idea/download/#section=windows>

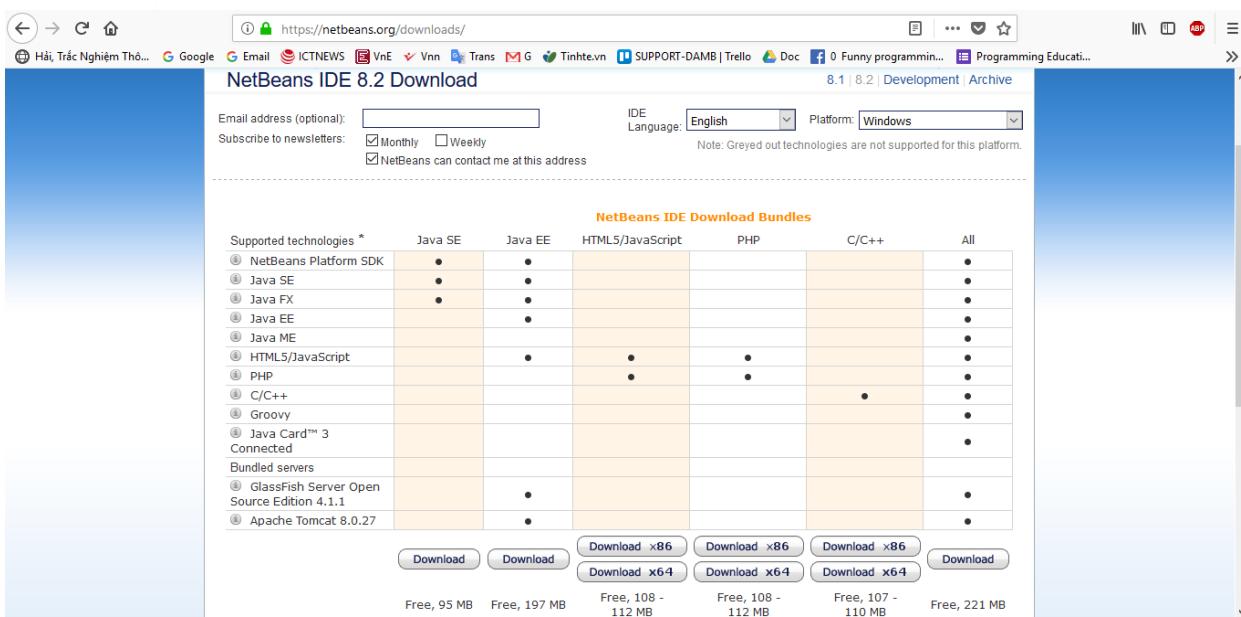
Nếu chỉ nghiên cứu Java thì chúng ta có thể chỉ cần tới bản **Community**

License	Commercial	Open-source, Apache 2.0
Java, Kotlin, Groovy, Scala	✓	✓
Android	✓	✓
Maven, Gradle, SBT	✓	✓
Git, SVN, Mercurial, CVS	✓	✓
Detecting Duplicates	✓	
Perforce, TFS	✓	
JavaScript, TypeScript	✓	
Java EE, Spring, GWT, Vaadin, Play, Grails, Other Frameworks	✓	
Database Tools, SQL	✓	

Dưới đây là giao diện tạo dự án mới để chúng ta dễ hình dung hơn.



- ❖ **Netbean IDE:** Trình soạn thảo khá phổ biến được hãng Oracle tài trợ phát triển, cũng chính là nhà tài trợ và phát triển các phiên bản Java hiện tại, dễ dùng hỗ trợ đầy đủ toàn bộ về Java (JavaSE, JavaEE, JavaFX) và bản Full sẽ hỗ trợ một số ngôn ngữ khác như C++, HTML, PHP, JavaScript đồng sẽ hỗ trợ cài đặt một webserver Glassfish cho các bạn học lập trình Java Web. Và hoàn toàn Free.
- **Link Download :** <https://netbeans.org/downloads/>

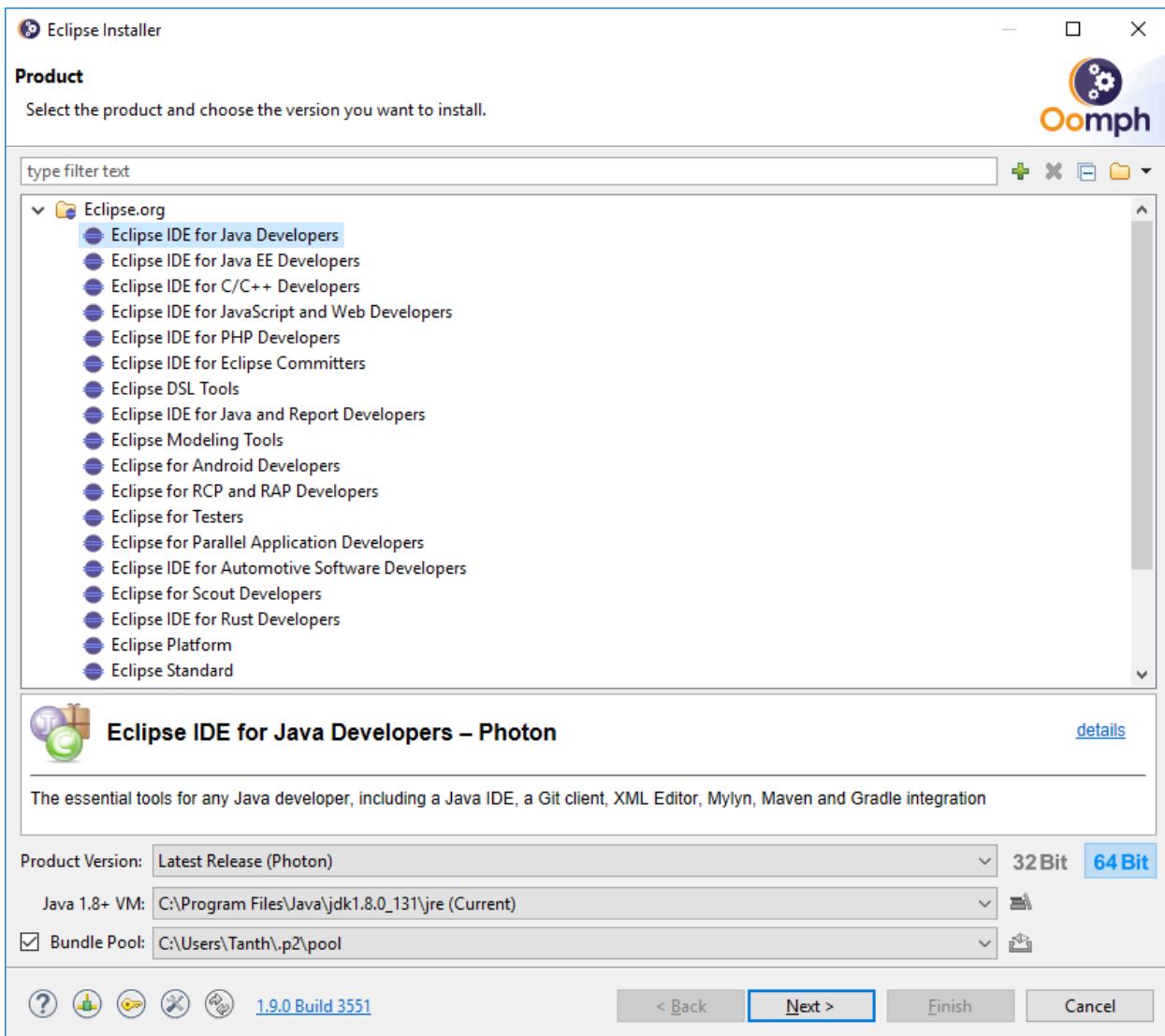


Dưới đây là giao diện tạo dự án mới để chúng ta dễ hình dung hơn.

- ❖ **Eclipse IDE:** Là một trong những IDE phổ biến và dễ dùng được phát triển bởi hãng IBM. So sánh với Netbean thì là kẻ tám lạng người nửa cân, đôi khi các lập trình viên dùng cả 2 IDE này tùy từng dự án và chúng cho phép import dự án của nhau vào nên rất thuận tiện.

Eclipse hỗ trợ đầy đủ Java giống như Netbean và các ngôn ngữ khác như C++, PHP, JavaScript, hỗ trợ phát triển cho Android, hỗ trợ Maven, Tuy nhiên quá trình cài đặt IDE này có vẻ phức tạp hơn và thiếu tính nhất quán do cài đặt các plugin

➤ Link download: <https://www.eclipse.org/downloads/>



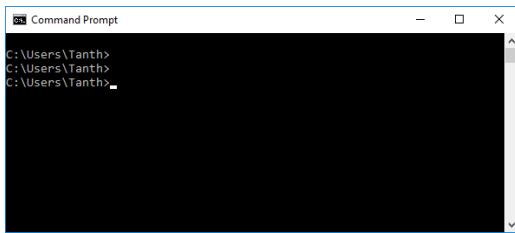
## 2.5 Lập trình chương trình Java đầu tay.

### 2.5.1 Tìm hiểu qua về chức năng bộ JDK một chút

- Javac: Là trình biên dịch chương trình mã nguồn (.java) thành tập tin byte code (.class)
- Java: Trình thông dịch java, dùng để thi hành chương trình java application
- Appletviewer: Trình duyệt applet, dùng để thi hành chương trình java applet

### 2.5.2 Tìm hiểu về các ứng dụng Java

- **Ứng dụng dòng lệnh (console Application):** Một loại chương trình không sử dụng giao diện đồ họa. khi chạy sẽ chỉ có cửa sổ console để giao tiếp với người dùng. Ứng dụng dạng này chủ yếu là core service hoặc ứng dụng tiện ích chạy dòng lệnh hoặc nhằm mục đích để học Java vì nó đơn giản.

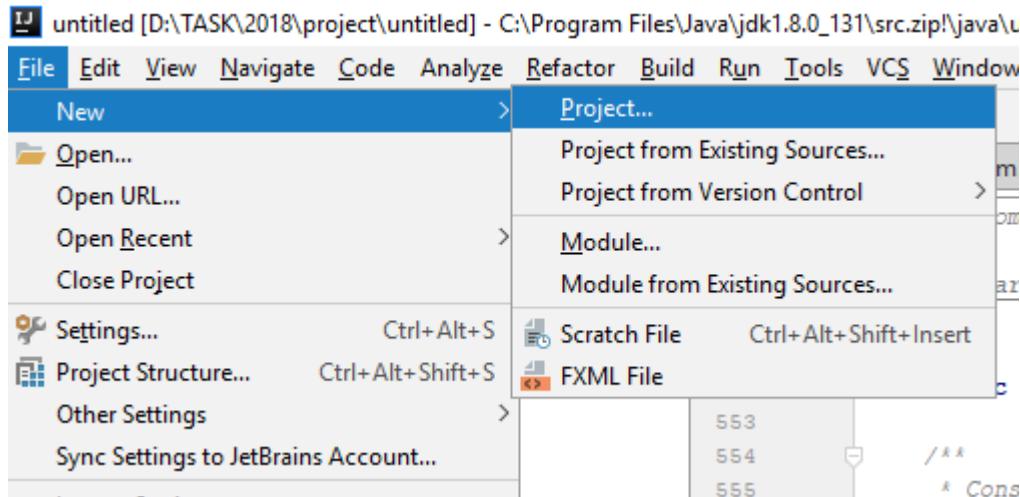


- **Ứng dụng đồ họa (graphics)** : là các dạng ứng dụng mục đích xây dựng cho desktop, có giao diện người dùng và giao tiếp với người dùng qua các cửa sổ form đồ họa. Ứng dụng này sử dụng các framework như JavaFX, swing(Đã cũ rồi)
- **Servlet**: Là loại ứng dụng server web, bạn chỉ cần học servlet ở mức độ biết là đủ, không cần đi quá sâu. Vì hiện chúng ta nên sử dụng các framework mạnh mẽ như Spring, Struts để tạo ra một trang web bằng java.
- **Applets**: Ứng dụng này là các chương trình Java chạy trên các Browser nhờ có Plugin Java Applets. Ứng dụng dạng này đã bị chính Oracle khai tử vì lý do bảo mật.

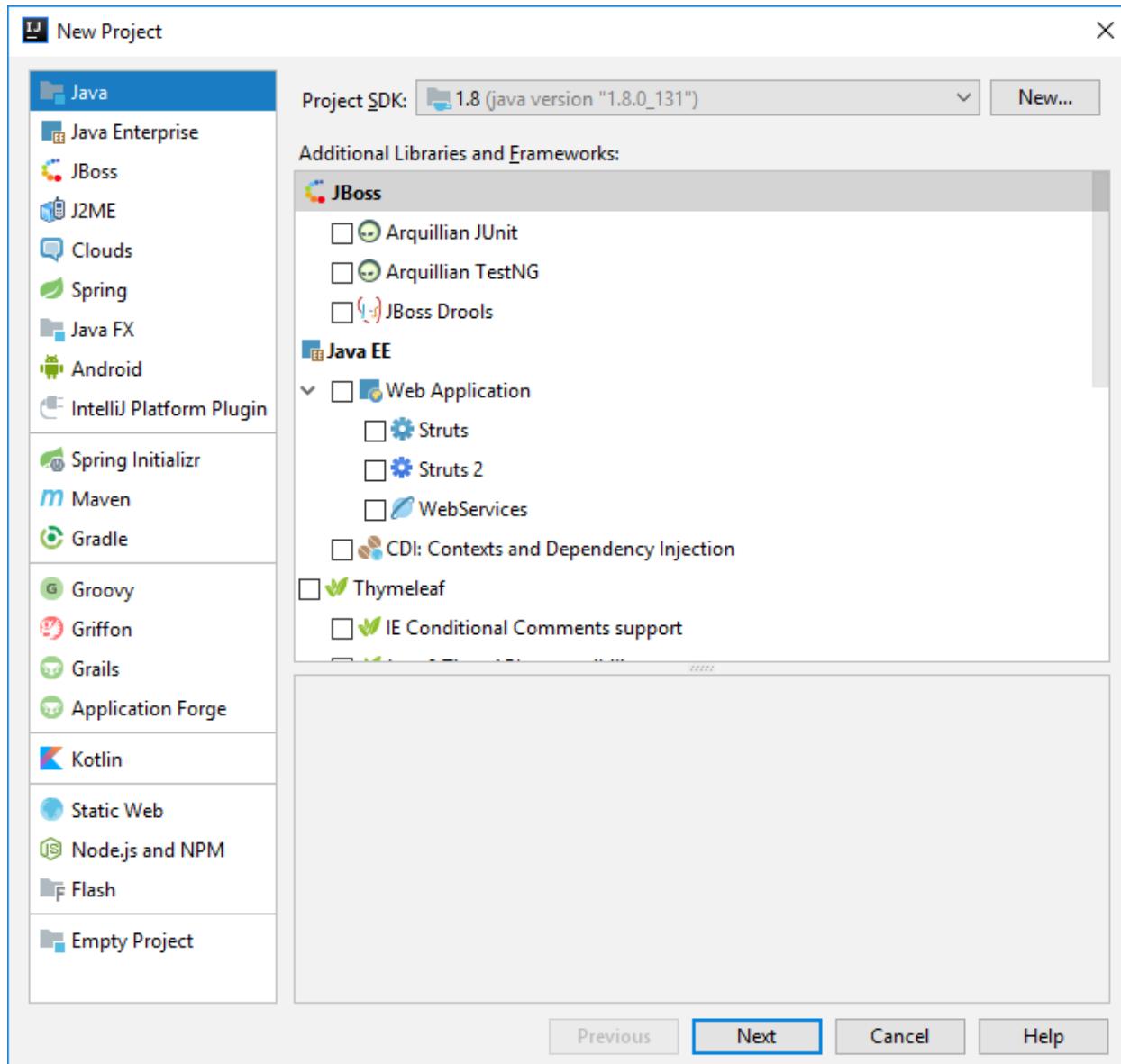
### 2.5.3 Tạo ứng dụng trên IDE

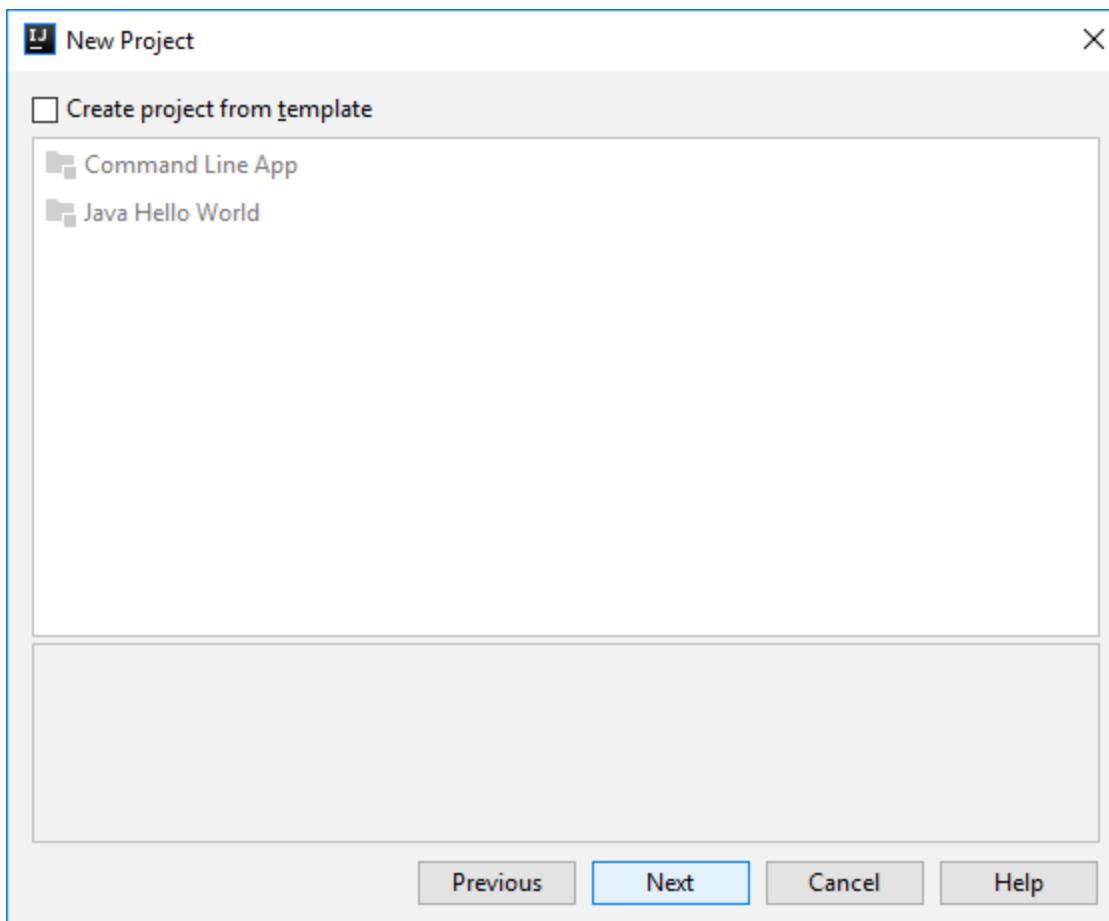
Chúng ta tạm lựa chọn Intelij IDE để làm mẫu. và các IDE khác cũng gần như tương tự.

- Bước 1: Mở IDE và tạo Project mới

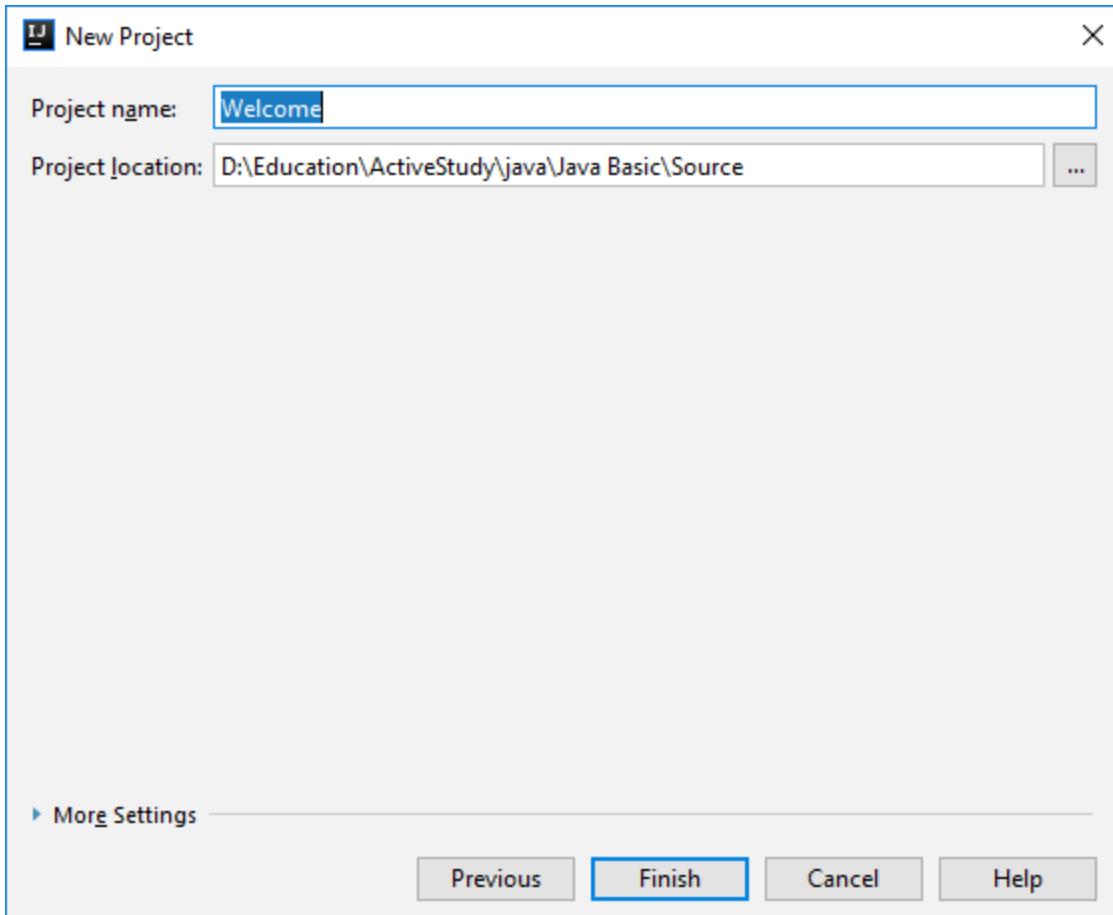


**Bước 2: Chọn loại Project Java:** Hiện tại chúng ta đang tạo một Project dạng Application đơn giản vì vậy chúng ta lựa chọn Java và không cần thêm các thư viện và Frameworks khác. Các bạn chú ý chọn SDK phiên bản phát triển cho phù hợp.

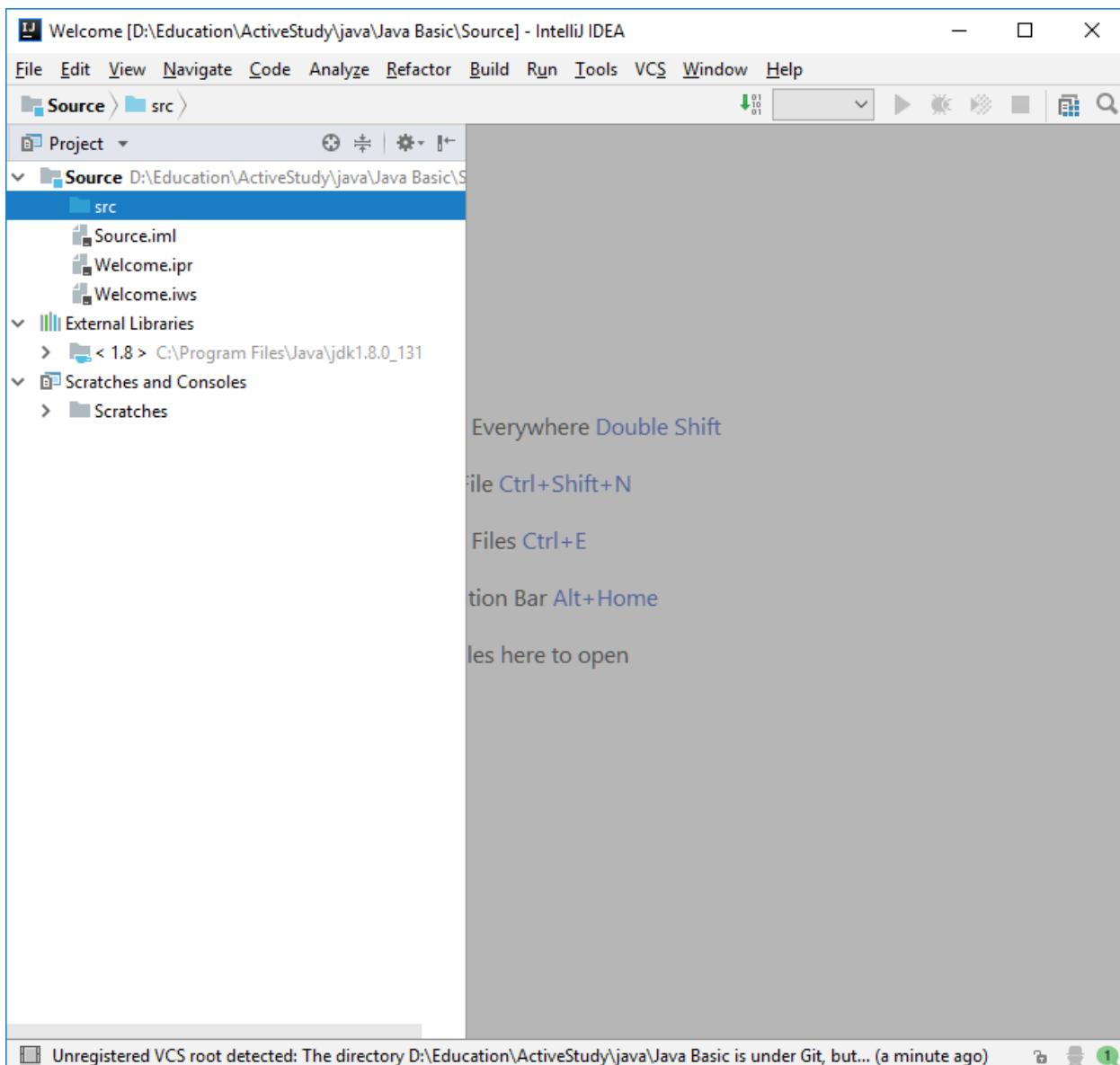




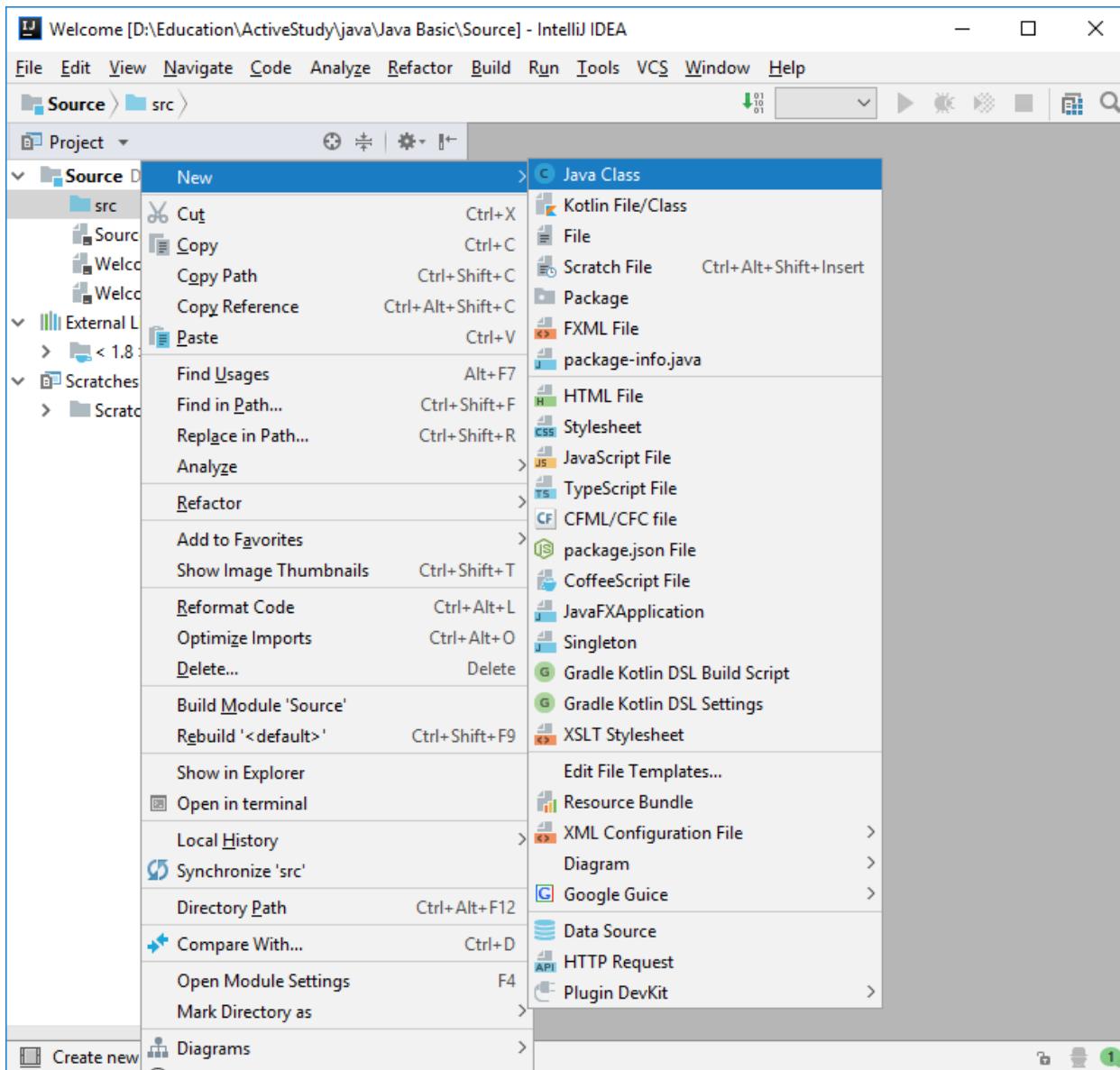
Bước 3: Nhập tên Project và chọn thư mục đường dẫn lưu trữ Project và mã nguồn



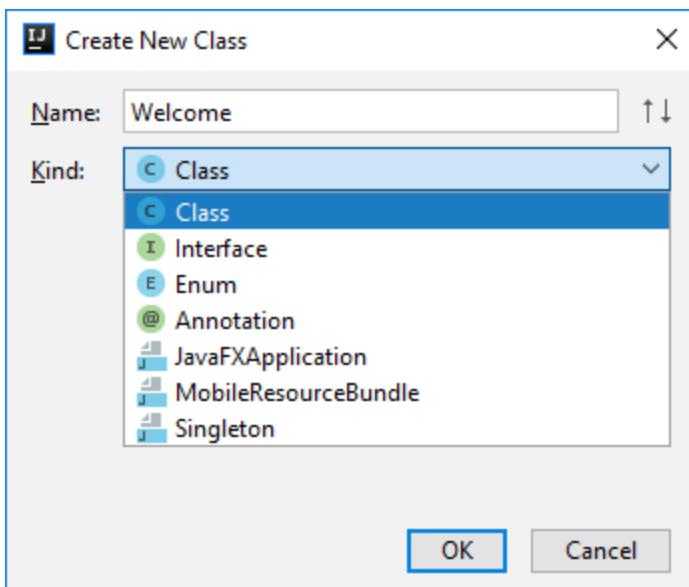
Project mới có tên Welcome đã được tạo ra và sẵn sàng để viết code



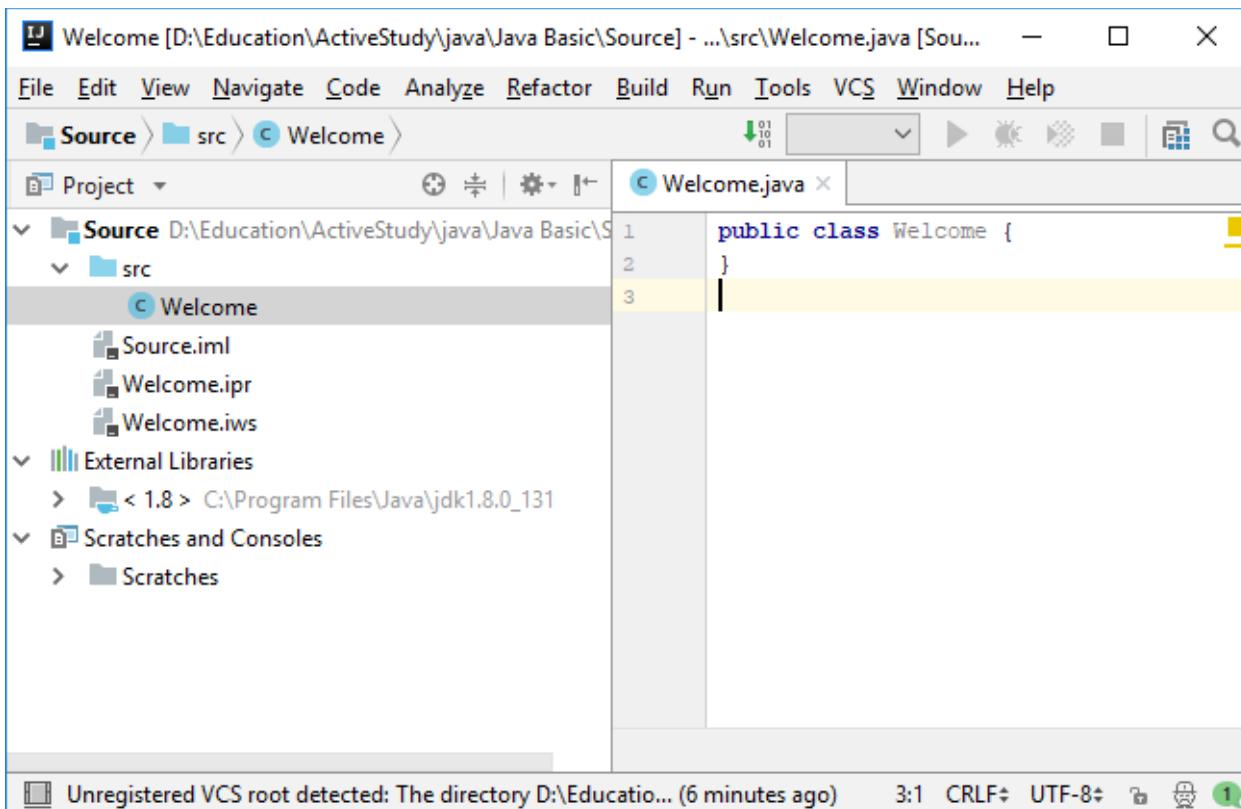
Bước 4: Thêm code vào Project “Welcome” bằng cách kích phải chuột vào thư mục “src” trong “source”



Chọn lựa class muốn thêm, ở đây ta chọn “class”



Class “Welcome” đã được tự động tạo ra và trùng tên với file “Welcome.java” việc tạo ra một file riêng và trùng tên với class ta đặt là một trong những quy tắc của Java chúng ta phải tuân theo.



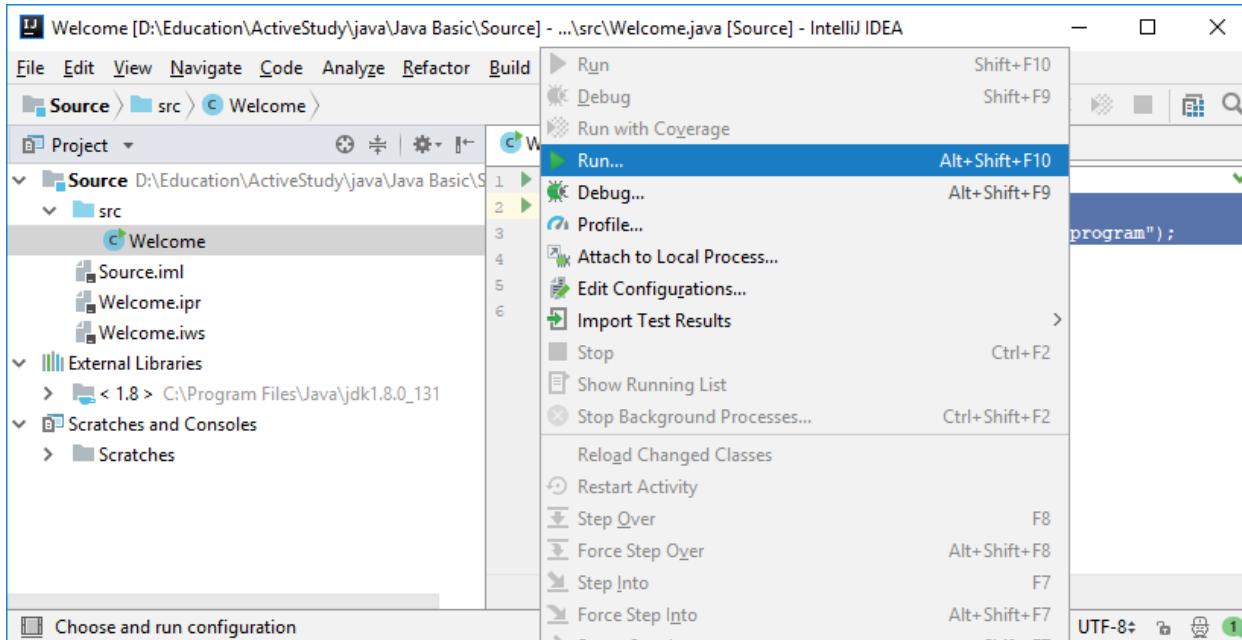
Bước 5 :Bây giờ chúng ta viết code vào class “Welcome” và chạy thôi nào

The screenshot shows the IntelliJ IDEA interface with the following details:

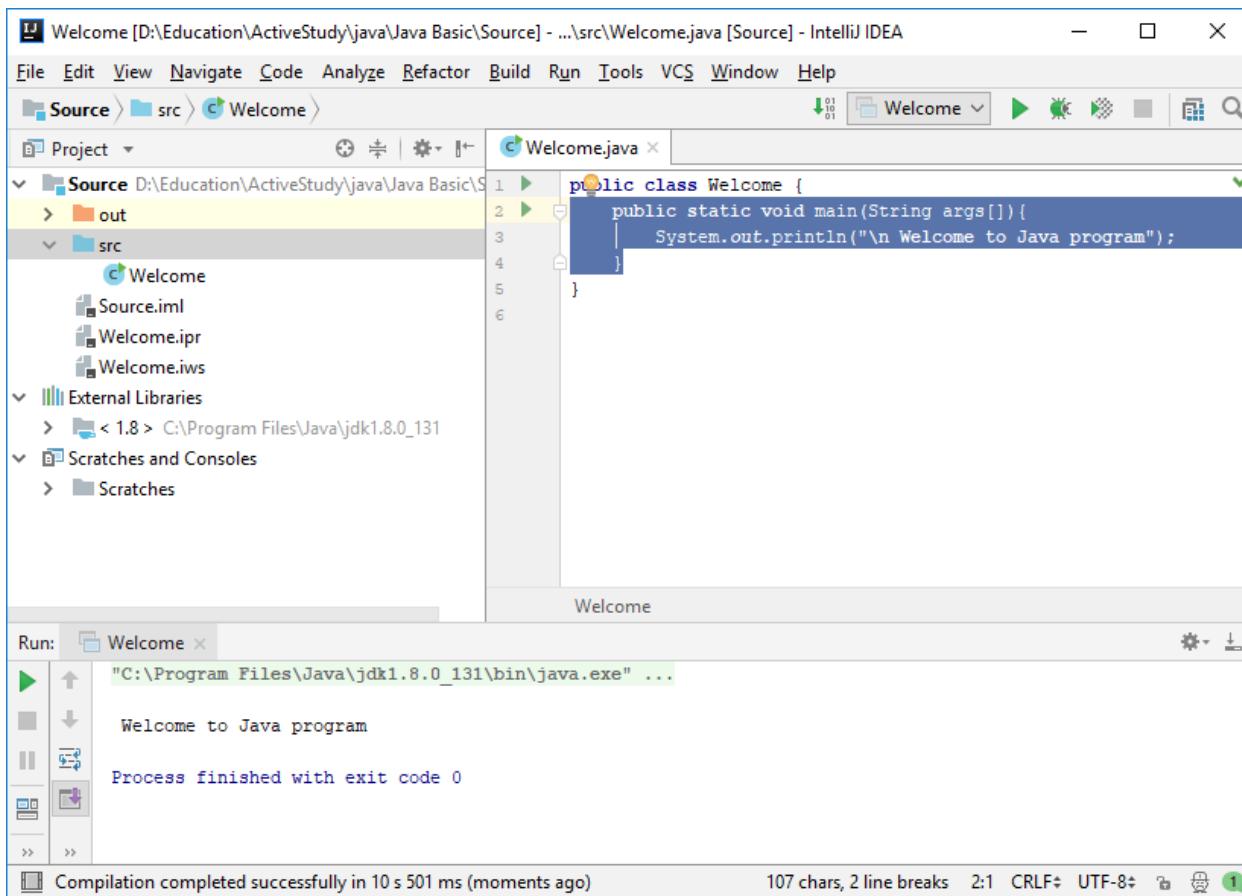
- Title Bar:** Welcome [D:\Education\ActiveStudy\java\Java Basic\Source] - ...\\src\\Welcome.java [Source] - IntelliJ IDEA
- Menu Bar:** File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
- Toolbars:** Standard, Version Control, Search, and others.
- Project Tool Window:** Shows the project structure under 'Source' with 'src' and 'Welcome' folders containing 'Source.iml', 'Welcome.ipr', and 'Welcome.iws' files.
- Content Area:** Displays the code editor for 'Welcome.java' with the following content:

```
1 public class Welcome {
2     public static void main(String args[]){
3         System.out.println("\n Welcome to Java program");
4     }
5 }
```
- Bottom Status Bar:** Unregistered VCS root detected: The directory D:\Education\Acti... (10 minutes ago) | 107 chars, 2 line breaks | 2:1 CRLF UTF-8 | Other icons.

Biên dịch và chạy project bằng cách truy cập menu “Run” và chọn “Run” hoặc dùng tổ hợp phím tắt “Alt Shift F10”



Kết quả sau khi chạy chương trình được hiển thị bên dưới



Vậy là chúng ta đã hoàn thành việc tạo và chạy một Project đầu tiên bằng IDE IntelliJ, khá là đơn giản và thuận tiện đúng không nào, tiếp theo thì chúng ta sẽ tiếp tục tiềm hiểu và làm nhiều project về Java nhé

## 2.6 Tìm hiểu nhanh các Khái niệm cần biết đầu tiên trước khi lập trình Java

- ❖ **Class (Lớp):** Là một kiểu dữ liệu có sẵn hoặc do lập trình viên tự thiết kế và cũng chính là bản thiết kế của các đối tượng (Object), trong mỗi class có các thuộc tính(variable) và hành vi (Method) thể hiện chức năng của đối tượng.

Ở bước tạo project đầu tiên chúng ta đã tự tạo ra một class của riêng chúng ta như bên dưới:

```

public class Person {
    String name;
    int high;
    int weight;
    int yearOfBirth;
    public void speak(){
        System.out.println("Hello, I am robot, my name is sophia.");
    }
}

```

- ❖ **Object** (Đối tượng): Là các thực thể được khai báo từ các class và được đại diện bởi các biến đối tượng (còn gọi là biến Reference).
- ❖ **Inheritance**: Kế thừa là một trong những thuộc tính của lập trình hướng đối tượng, Các lớp con được phép kế thừa các thuộc tính và hành vi từ lớp cha, tuy nhiên không phải lớp con sẽ kế thừa toàn bộ mà sẽ theo quy định của lớp cha cho phép , điều này chúng ta hoàn toàn có thể liên tưởng tới việc con cái trong gia đình dòng họ kế thừa gen và tài sản của cha mẹ ông bà, việc kế thừa là mặc định theo luật định nếu như không có di chúc của ông bà cha mẹ để lại, chúng ta sẽ tìm hiểu sâu hơn về vấn đề này khi tìm hiểu về OOP (Tính chất hướng đối tượng)

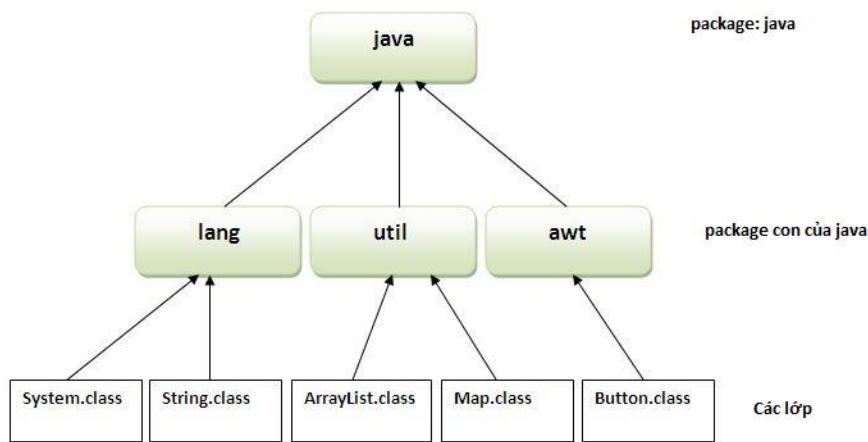
```

public class InheritanceClass extends Welcome{
}

```

- ❖ **Access Modifier** : Từ khóa dùng trong Class để quy định phạm vi truy cập đối với thuộc tính và hành vi của một đối tượng ví dụ như : public, protected, private. Chúng ta sẽ tìm hiểu sâu về vấn đề này ở các nội dung phía sau, hiện giờ chúng ta nắm bắt để không bị bỡ ngỡ đã nhé
- ❖ **Package**: Là một đơn vị (Thành phần) để chứa đựng (đóng gói) các file code Java. Các Package có thể chứa các Package con bên trong.
  - Package được sử dụng để phân loại lớp và interface giúp dễ dàng bảo trì. Các lớp, Interface có cùng đặc điểm hoặc chức năng chung thì được chứa đựng vào cùng một Package tùy theo tổ chức của lập trình viên.
  - Package tạo ra một phạm vi về truy cập, sự truy cập đến các lớp, thuộc tính và hành vi của đối tượng chịu sự chi phối của các Access Modifier và Package.

- Package khắc phục được việc đặt trùng tên.



## 2.7 Chú ý các quy tắc trong viết code Java

### 2.7.1 Quy tắc đặt tên trong Java

- Đặt tên là một trong những kỹ năng quan trọng đối với lập trình viên, ngoài việc đảm bảo tính trong sáng, ý nghĩa và gợi nhớ thì cần phải đảm bảo theo quy tắc nhất quán để tất cả mọi người đều có thể đọc và bảo trì (maintain) code một cách dễ dàng.

#### 2.7.1.1 Quy tắc chung

- ❖ Nên khai báo tên có ý nghĩa và thể hiện được mục đích, dễ hiểu
- ❖ Tên khai báo không nên dài quá 20 ký tự hoặc có thể ít hơn nhưng phải đảm bảo đầy đủ về mặt ý nghĩa của nó, và tên cũng không được đặt quá ngắn, trừ khi đó là tên tạm  
**Example:** `a, i, j,...`
- ❖ Tránh đặt những tên tương tự nhau. **Example:** hai biến có tên là `personOne` và `person1` không nên được sử dụng trong một Class vì sẽ dễ gây ra nhầm lẫn trong quá trình viết code.
- ❖ Hạn chế viết tắt trừ khi từ viết tắt đó phổ biến và được nhiều người biết đến. **Example:** Chúng ta không nên đặt tên biến là `soTB` mà nên đặt tên là `soThueBao`. nhưng lại hoàn toàn có thể đặt tên là `fileHTML` thay vì `fileHypertextMarkupLanguage` vì tên này quá dài và từ HTML cũng là một từ khá phổ biến, ít nhất là trong giới lập trình viên chúng ta.
- ❖ Tên các biến cục bộ của lớp nên kết thúc bằng hậu tố “\_” **Example:** `count_`

- ❖ Tất cả các tên nên được viết bằng Tiếng Anh, Tránh kết hợp nhiều ngôn ngữ khác nhau (*Tiếng Anh + Tiếng Việt + ...*), **Example:** addPerson...
- ❖ Những biến phạm vi rộng nên đặt tên dài, những biến phạm vi hoạt động hẹp (cục bộ) nên đặt tên ngắn.
- ❖ Từ khóa “set/get” phải được đặt trong các phương thức truy cập trực tiếp đến thuộc tính: **Example:** getName(), setSalary(int)...
  
- ❖ Không trùng với các “*từ khóa*”
- ❖ Không được bắt đầu bằng số, **Example:** 123Person.
- ❖ Tên phải được bắt đầu bằng một chữ cái, hoặc các ký tự như \$, \_, ...
- ❖ Không được chứa khoảng trắng, các ký tự toán học. Nếu tên bao gồm nhiều từ thì phân cách nhau bằng dấu \_.
- ❖ Trong Java có phân biệt chữ hoa chữ thường. **Example:** như hocSinh sẽ khác với hoCSinh.
  
- ❖ Tiền tố “is” nên được sử dụng trong các phương thức, các biến kiểu boolean:

**Example:** isEmpty, isOpen...

- ❖ Từ khóa “find” có thể được sử dụng trong các phương thức tìm kiếm:

**Example:** person.findFriend();

- ❖ Những phần bổ sung thêm nên được sử dụng nhằm chỉ rõ hành động, mục đích hoặc tính chất:

**Example:** get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc.

### 2.7.1.2 Đặt tên biến và hằng số

- ❖ Biến (Variable) nên được đặt theo **quy tắc lạc đà** (*Camel Case*) bắt đầu bằng một ký tự in thường, các từ tiếp theo được bắt đầu bằng một ký tự in hoa **Example:** yearOfBirth

- ❖ Ngoài ra, trong một số trường hợp, tên biến cần phải thể hiện rõ kiểu dữ liệu của biến đó.  
**Example:** biến có kiểu là List thì nên đặt tên là studentList, biến có kiểu là Set thì nên đặt tên là studentSet, biến có kiểu là Map thì nên đặt tên là studentMap, biến có kiểu là Array thì nên đặt tên là studentArray,...
- ❖ Hằng (Constant) phải đặt toàn bộ là chữ in hoa, các từ tách biệt nhau bởi ký tự gạch dưới “\_”. **Example:** TOI\_LA\_CONST
- ❖ Các biến JFC (Java Swing) nên được đặt hậu tố là kiểu đối tượng: **Example:** widthScale, nameTextField, leftScrollbar, mainPanel, fileToggle, minLabel, printerDialog
- ❖ Tập hợp nhiều đối tượng nên được đặt tên ở số nhiều: **Example:** Collection< Person > persons;
- ❖ Những biến chỉ số lượng đối tượng nên có tiền tố “n”: Example: nCounter, nPerson...

#### 2.7.1.3 *Đặt tên phương thức (Method).*

- ❖ Phương thức (Method) nên là động từ, bắt đầu bằng 1 ký tự in thường và các từ tiếp sau được bắt đầu bằng 1 ký tự in hoa cũng theo quy tắc lạc đà (*Camel Case*)

#### 2.7.1.4 *Đặt tên Class và Interface.*

- ❖ Class nên được đặt tên là danh từ, và bắt đầu các từ bằng 1 ký tự in hoa **Example:** Person
- ❖ Tên Interface nên có thêm chữ I đằng trước. **Example:** IFrame.
- ❖ Tên lớp trừu tượng (dẫn xuất) nên có từ Abstract làm tiền tố,  
**Example:** AbstractAnimal (chúng ta sẽ tìm hiểu về lớp dẫn xuất ở những bài sau).

#### 2.7.1.5 *Đặt tên Package.*

- ❖ Package nên được đặt tên bằng toàn bộ chữ tên thường **Example:** jb12

#### 2.7.1.6 *Đặt tên Project.*

- ❖ Tên Project phải tuân theo quy tắc chung ở trên và chữ cái đầu tiên của mỗi từ phải viết hoa.

## 2.7.2 Comment(Ghi chú) code trong Java

- Trong quá trình viết code chúng ta cần giải thích về chức năng nhiệm vụ và ý nghĩa của các đoạn code, của một phương thức hay class... hoặc để tạm bỏ một dòng code. Trong quá trình biên dịch thì trình biên dịch Javac sẽ bỏ qua tất cả các comment code này và chỉ dịch các đoạn code của chúng ta thôi. Trong Java chúng ta có 3 cách comment code như sau:

Comment	Miêu tả
// text	<p>Comment một dòng. Trình biên dịch sẽ bỏ qua mọi thứ từ // tới cuối dòng.</p> <p>Thường được dùng để che lại một đoạn code hoặc mô tả nhanh về một dòng code</p>
/* text */	<p>Comment nhiều dòng. Trình biên dịch bỏ qua mọi thứ từ /* tới */.</p> <p>Thường được dùng để mô tả về chức năng nhiệm vụ của một đoạn code, một method hoặc class.</p>
/** documentation */	<p>Sử dụng Javadoc. Khi comment kiểu này trình Javadoc (thuộc bộ JDK) sẽ hiểu và có thể xuất ra các file document dạng html/css giúp làm tài liệu của file Java và dự án.</p> <p>Phương pháp này được khuyến cáo sử dụng trong các dự án chuyên nghiệp giúp tạo ra hệ thống document của dự án chi tiết đầy đủ và nhanh gọn.</p> <p>Thường được sử dụng để mô tả chức năng nhiệm vụ của một Class, Method.</p>

- Ví dụ comment một dòng

```
public class Welcome {
    public static void main(String args[]){
        //The line below is used to print the text to the console
        System.out.println("Welcome to Java program");
    }
}
```

- Ví dụ comment nhiều dòng

```
public class Welcome {
    /*This is main method.
     A Java program can not do without it
    */
    public static void main(String args[]){
        System.out.println("Welcome to Java program");
    }
}
```

- Ví dụ dùng Javadoc.

```
/**
 * @author Tanth
 * @version 1.0
 * @apiNote for Hello world project
 * @deprecated
 * @implNote
 * @implSpec
 * @param
 * @see
 * @serial
 * @see
 */
public class Welcome {
    public static void main(String args[]){
        System.out.println("\n Welcome to Java program");
    }
}
```

- Khi chúng ta gõ chữ **@** trong phần viết comment Javadoc thì IDE nó sẽ hiện ra 1 loạt các tag như hình trên. Các tag này sẽ giúp chúng ta giải thích 1 cách rõ ràng nhất về tham số, giá trị trả về ...

```

1  /**
2   * @author Tanth
3   * @version 1.0
4   * @apiNote for Hello world project
5   * @deprecated
6   * @implNote
7   * @implSpec
8   * @param
9   * @see
10  * @serial
11  */
12  public class Welcome {
13      /**
14       * @see serial
15       * @param param
16       * @implSpec
17       * @implNote
18       * @deprecated
19       * @apiNote
20       * @version
21       * @author
22       * @author Tanth
23       * @since
24   }

```

- Sau đây chúng ta sẽ nghiên cứu một số tag sử dụng vào các chức năng chuyên biệt như bên dưới đây.

Thẻ	Miêu tả	Cú pháp
@author	chú thích tên tác giả viết đoạn code này	@author name-text
{@code}	Hiển thị text trong code font mà không thông dịch text như là các thẻ HTML markup hoặc javadoc được lồng.	{@code text}

{@docRoot}	Biểu diễn relative path tới thư mục root của tài liệu từ các trang được tạo	{@docRoot}
@deprecated	Chú thích chỉ dẫn rằng API/Method này không còn được sử dụng nữa	@deprecated deprecated-text
@exception	Chú thích về một Exception có thể được <b>Throws</b> ra với tên lớp và text mô tả	@exception class-name Miêu tả
{@inheritDoc}	Kế thừa một comment từ lớp có thể kế thừa <b>gần nhất</b> hoặc interface có thể triển khai	Kế thừa một comment từ một lớp cha trung gian.
{@link}	Chèn một in-line link với nhãn text có thể nhin thấy mà chỉ tới Documentation cho package, lớp hoặc tên member đã xác định của một lớp tham chiếu	{@link package.class#member label}
{@linkplain}	Giống {@link}, ngoại trừ label của link được hiển thị ở dạng plain text thay vì code font.	{@linkplain package.class#member label}
@param	Thêm một tham số với parameter-name đã xác định được theo sau bởi description đã xác định tới khu vực "Parameters".	@param parameter-name Miêu tả
@return	Thêm một khu vực "Returns" với Miêu tả	@return Miêu tả
@see	Thêm một "See Also" heading với một link hoặc text mà chỉ tới reference	@see reference

@serial	Được sử dụng trong doc comment cho một trường có thể xếp thứ tự.	@serial field-description   include   exclude
@serialData	Thu thập thông tin dữ liệu được viết bởi phương thức writeObject( ) hoặc writeExternal( )	@serialData data-Miêu tả
@serialField	Thu thập thông tin một thành phần ObjectStreamField	@serialField field-name field-type field-Miêu tả
@since	Thêm một "Since" heading với since-text đã xác định tới Documentation đã tạo	@since release
@throws	Thẻ @throws và @exception là như nhau	@throws class-name Miêu tả
{@value}	Khi {@value} được sử dụng trong doc comment của một trường static, nó hiển thị giá trị của hằng số đó	{@value package.class#field}
@version	Thêm một "Version" subheading với version-text đã xác định tới Documentation đã tạo khi tùy chọn - version được sử dụng	@version version-text

### 2.7.3 Các từ khóa quan trọng trong Java.

- Ngôn ngữ [lập trình Java](#) có khoảng 50 keywords, Chúng ta cần làm quen và tóm những keyword này để không bị bỡ ngỡ và hoang mang khi gặp chúng trong quá trình đọc code mẫu hoặc lúc lập trình.
- Lưu ý: true, false, null không phải là các keyword, chúng là các giá trị. Dưới đây là danh sách keyword trong Java:

Keyword	What It Does
abstract	Khai báo lớp, phương thức, interface trừu tượng không có thể hiện(instance) cụ thể
Boolean	Khai báo biến kiểu logic với 2 trị: true, false.
Break	Thoát ra khỏi vòng lặp hoặc lệnh switch-case.
byte	Kiểu byte với các giá trị nguyên chiếm 8 bit (1 byte).
case	Trường hợp được tuyển chọn theo switch (chỉ được dùng khi đi kèm switch)
catch	Được sử dụng để bắt ngoại lệ, được sử dụng cùng với try để xử lý các ngoại lệ xảy ra trong chương trình
char	Kiểu ký tự Unicode, mỗi ký tự chiếm 16 bit (2 byte).
class	Được sử dụng để định nghĩa class
continue	Dừng chu trình(iteration) lặp hiện tại và bắt đầu chu trình tiếp theo
default	Mặc định được thực thi khi không có case nào trả về giá trị true (dùng trong switch case)
do	Dùng trong vòng lặp do while
double	Kiểu số thực với các giá trị biểu diễn theo dạng dấu phẩy động 64 bit (8 byte).
else	Rẽ nhánh theo điều kiện ngược lại của if.
enum	Định nghĩa kiểu dữ liệu enum - gần giống với kiểu dữ liệu mảng nhưng các phần tử có thể bổ sung thêm các phương thức

<code>extends</code>	Được sử dụng để định nghĩa lớp con kế thừa các thuộc tính và phương thức từ lớp cha.
<code>final</code>	Chỉ ra các biến, phương thức không được thay đổi sau khi đã được định nghĩa. Các phương thức final không thể được kế thừa và override
<code>finally</code>	Thực hiện một khối lệnh đến cùng bất chấp các ngoại lệ có thể xảy ra. Được sử dụng trong try-catch
<code>float</code>	Kiểu số thực với các giá trị biểu diễn theo dạng dấu phẩy động 32 bit.
<code>for</code>	Sử dụng cho vòng lặp for với bước lặp được xác định trước
<code>If</code>	Lệnh chọn theo điều kiện logic
<code>implements</code>	Xây dựng một lớp mới cài đặt những phương thức từ interface xác định trước.
<code>import</code>	Yêu cầu một hay một số lớp ở các gói chỉ định cần nhập vào để sử dụng trong ứng dụng hiện thời.
<code>instanceof</code>	Kiểm tra xem một đối tượng nào đó có phải là một thể hiện của 1 class được định nghĩa trước hay không
<code>int</code>	Kiểu số nguyên với các giá trị chiếm 32 bit (4 byte).
<code>interface</code>	Được sử dụng để định nghĩa interface
<code>long</code>	Kiểu số nguyên lớn với các giá trị chiếm 64 bit (8 byte).
<code>native</code>	Giúp lập trình viên có thể sử dụng code được viết bằng các ngôn ngữ khác
<code>new</code>	Khởi tạo đối tượng

package	Xác định một gói sẽ chứa một số lớp ở trong file mã nguồn.
private	Khai báo biến dữ liệu, phương thức riêng trong từng lớp và chỉ cho phép truy cập trong lớp đó.
protected	Khai báo biến dữ liệu, phương thức chỉ được truy cập ở lớp cha và các lớp con của lớp đó.
public	Khai báo lớp, biến dữ liệu, phương thức công khai có thể truy cập ở mọi nơi trong hệ thống.
return	Kết thúc phương thức và trả về giá trị cho phương thức
short	Kiểu số nguyên ngắn với các giá trị chiếm 16 bit (2 byte).
static	Định nghĩa biến, phương thức của một lớp có thể được truy cập trực tiếp từ lớp mà không thông qua khởi tạo đối tượng của lớp
super	Biến chỉ tới đối tượng ở lớp cha
switch	Sử dụng trong câu lệnh điều khiển switch case
synchronized	Thường được sử dụng trong lập trình đa luồng (multithreading), Từ khóa để nhằm đồng bộ giữa các tiểu tiến trình tránh xung đột khi sử dụng chung các đối tượng tài nguyên như đối tượng, đoạn code, method...
this	Biến "this" chỉ tới đối tượng hiện thời.
throw	Tạo một đối tượng exception để chỉ định một trường hợp ngoại lệ xảy ra. Khi throw ra thì cần phải catch ở ngoài để đảm bảo chương trình không bị dừng đột ngột.

throws	Chỉ định cho qua ngoại lệ khi exception xảy ra, sử dụng ở một method khi khai báo method.
transient	Chỉ định rằng nếu một đối tượng được serialized, giá trị của biến sẽ không cần được lưu trữ
try	Bắt đầu của một đoạn code catch ngoại lệ.
void	Chỉ định một phương thức không trả về giá trị
volatile	Báo cho chương trình dịch biết là biến khai báo volatile có thể thay đổi tùy ý trong các luồng (thread).
while	Được sử dụng trong lệnh điều khiển while
assert	Kiểm tra điều kiện đúng hay sai (thường dùng trong Unit Test)

## 2.8 Variables (Biến số) và Kiểu dữ liệu trong Java

### 2.8.1 Biến và phạm vi biến trong Java

- Xét ví dụ về class Person sau:

```

7  /**
8  * @author tanth
9  */
10 public class Person {
11     String name;
12     int high;
13     int weight;
14     int yearOfBirth;
15     public void speak(){
16         System.out.println("Hello, I am robot, my name is sophia.");
17     }
18 }
```

- Trong ví dụ trên ta có các biến sau:

```

11     String name;
12     int high;
13     int weight;
14     int yearOfBirth;

```

- Biến hay còn gọi là thuộc tính là một thành phần được khai báo để chứa dữ liệu. Các giải thuật sẽ xử lý các dữ liệu này tùy theo yêu cầu nghiệp vụ.
- Mỗi một biến đều được khai báo trong một class, method hoặc một blog code nào đó.
- Biến có tên để sử dụng và có kiểu dữ liệu được định trước khi khai báo và được cấp một vùng nhớ trên bộ nhớ khi chạy để lưu trữ dữ liệu.
- Cú pháp khai báo biến như sau: “**Kiểu Dữ Liệu**” “**Tên Biến**” Ví dụ: **int yearOfBirth;** Trong đó **int** là kiểu dữ liệu, **yearOfBirth** là tên biến, tên này được sử dụng để truy cập vào giá trị của biến hoặc gán lại giá trị cho biến. Khi ta khai báo biến **yearOfBirth** như ví dụ này thì có nghĩa là khai báo với máy tính hiểu rằng cấp phát một vùng nhớ tương ứng để chứa đựng dữ liệu kiểu **int**.
- Các biến có cùng kiểu dữ liệu thì có thể khai báo chung một dòng như sau:

```

18     int high, weight, yearOfBirth;
19

```

- Thông thường giá trị của biến cần phải được khởi tạo và ta có thể khởi tạo giá trị của một biến số trên cùng 1 dòng khi khai báo:

15      **int high = 170;**  
**int weight;**

- Các biến khi khai báo đều đã có giá trị mặc định ban đầu tùy vào kiểu dữ liệu của nó chúng ta cần phải nắm rõ điều này để tránh các lỗi logic không đáng có. Tuy vậy chúng ta cần phải có thói quen tự khởi tạo giá trị ban đầu để chắc chắn về giá trị của biến số trước khi đi xử lý trong giải thuật của chúng ta để giải thuật chạy được chính xác.
- Sở dĩ được gọi là biến vì giá trị của chúng có thể được thay đổi bằng phép gán **high = 170;**
- Để gán giá trị hoặc truy cập tới các biến thì chúng ta sử dụng tên biến để lấy giá trị biến. VD:

```

String say_ = "Hello, I am robot, my name is " + name;
System.out.println(say_);

```

Và kết quả sau khi chạy đoạn lệnh trên sẽ là:

Output - JavaCoreBasic (run) ×

```

run:
Hello, I am robot, my name is sophia.
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Khi gán hoặc truy cập tới biến chúng ta cần chú ý tới kiểu giá trị của biến. ví dụ ta có thể cộng một biến kiểu String với một biến kiểu số để ra một String, tuy nhiên hai biến kiểu số cộng với nhau sẽ ra một số tổng.

```

22 |     int sum = high + weight;
23 |     System.out.println("My high and weight is " + sum);

```

Kết quả thực hiện như sau:

Output - JavaCoreBasic (run) ×

```

run:
Hello, I am robot, my name is sophia.
My high and weight is 235
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Các biến khi khai báo tại các vị trí thì sẽ có phạm vi ảnh hưởng và truy cập khác nhau tạo ra các khái niệm biến Instance và biến Local.
  - o Các biến khai báo trong class (Instance variables) và nằm ngoài method thì sẽ có phạm vi tham chiếu trong class (Cả class và tất cả các method), nghĩa là ta có thể truy cập đến biến loại này từ bất cứ method nào trong class khai báo nó.

```

11  public class Person {
12
13
14     String name;
15     int high = 170; //Biến Instance
16     int weight = 65;
17     int yearOfBirth;
18
19     public void speak() {
20         System.out.println("Hello, I am robot, my name is sophia.");
21         int sum = high + weight; //sum là biến local
22         System.out.println("My high and weight is " + sum);
23     }
24 }

```

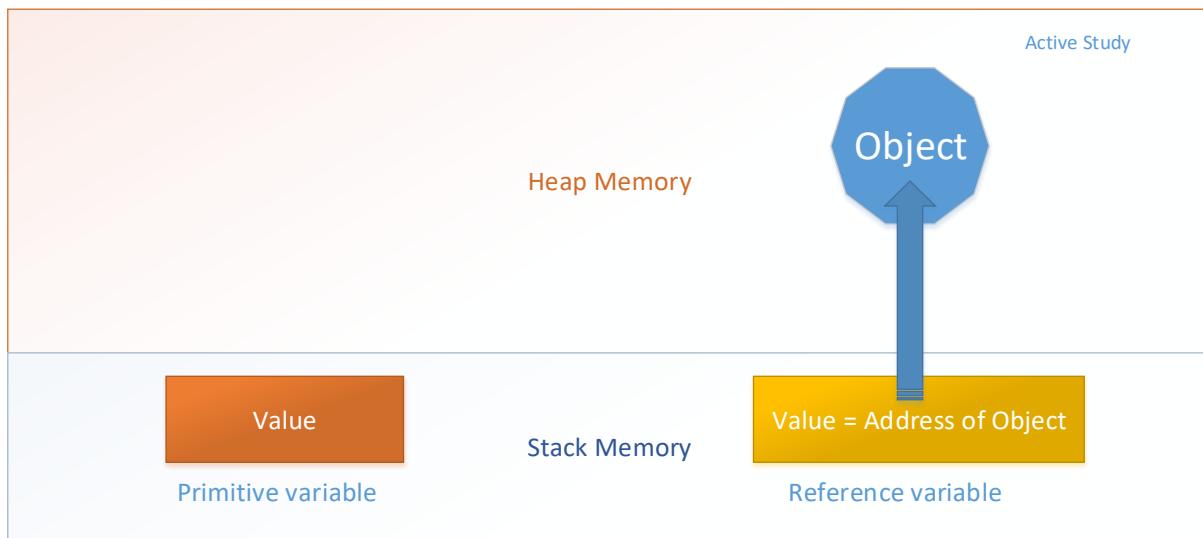
- o Các biến khai báo trong method (Local variables) thì có phạm vi tham chiếu trong method đó mà thôi, nghĩa là ta không thể truy cập tới biến này từ bên ngoài method khai báo nó.
- o Trong cùng một phạm vi không thể có 2 biến cùng tên, ở các phạm vi khác nhau thì điều này là bình thường, tuy nhiên khi hai phạm vi có sự chồng lấn lên nhau sẽ nảy sinh vấn đề nhầm lẫn. Ví dụ trong một class và method của class đó cùng có một biến có tên giống nhau (Có thể cùng hoặc khác kiểu dữ liệu) thì khi sử dụng biến cần chỉ ra biến đang sử dụng là của class hay của method. Ví dụ tại dòng 19

```

11  public class Person {
12      private String name = "sophia";
13      private int high;
14      private int weight;
15      private int yearOfBirth;
16      private boolean isMarried;
17      private final String TOI_LA_CONST = "Tôi là constant";
18      public void speakName(String name) {
19          this.name = name;
20          String say_ = "Hello, I am robot, my name is " + name;
21          System.out.println(say_);
22      }

```

- Chúng ta có 2 kiểu dữ liệu đó là kiểu nguyên thủy (Primitive) hay còn gọi là kiểu dữ liệu cơ bản (basic) và không nguyên thủy (nonPrimitive) , tương ứng với 2 loại kiểu dữ liệu này ta cũng có 2 loại biến chúng ta cần phân biệt được đó là biến nguyên thủy (Primitive) còn gọi đơn giản là biến (Variables) và biến tham chiếu (Reference variables) hoặc còn gọi là biến đối tượng (Object reference variables).



- Biến nguyên thủy nắm giữ giá trị dữ liệu còn biến tham chiếu nắm giữ địa chỉ của ô nhớ trong bộ nhớ nơi mà dữ liệu được lưu trữ.

VD biến nguyên thủy có tên high:

`int high = 170;`

VD biến tham chiếu có tên hocVien:

`Person hocVien = new Person();`

Khi xuất ra màn hình hai biến này sẽ thấy sự khác biệt:

The screenshot shows a Java code editor with a file named `Welcome.java`. The code defines a class `Welcome` with a `main` method. Inside the `main` method, a `Person` object is created with the name "Hieu", and its height is set to 170. The program then prints the value of `high` and the `toString` representation of the `Person` object. Below the code editor is a terminal window titled `: Output - JB12 (run)` showing the execution results.

```

11  */
12
13  public class Welcome {
14      public static void main(String[] args) {
15          Person hocVien = new Person("Hieu");
16          int high = 170;
17          System.out.println("high = " + high);
18          System.out.println("hocVien = " + hocVien);
19      }
20  }
21

```

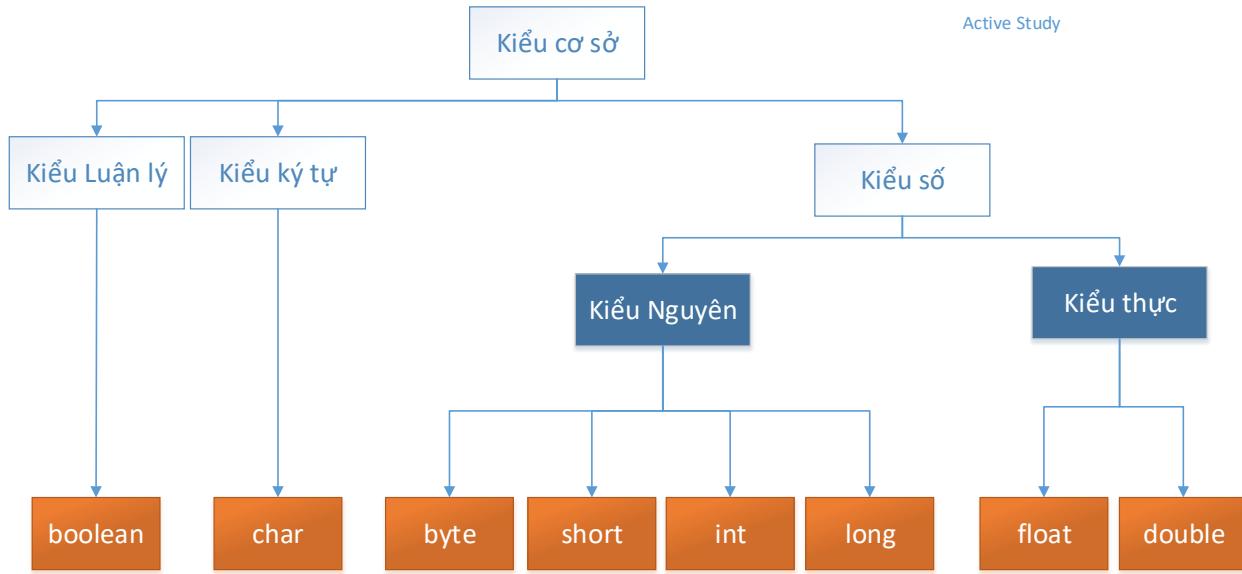
```

: Output - JB12 (run)
run:
high = 170
hocVien = jb12.Person@15db9742
BUILD SUCCESSFUL (total time: 0 seconds)

```

## 2.8.2 Các kiểu dữ liệu nguyên thủy trong Java

- Ngôn ngữ lập trình java có 8 kiểu dữ liệu nguyên thủy đó là : byte, short, int, long, float, double, boolean và char.



- Kích cỡ chiếm dụng bộ nhớ và miền giá trị của các kiểu dữ liệu được mô tả ở bảng sau:

Tên	Byte	Bit	Dấu (+ - )	Khoảng giá trị số mũ	Khoảng giá trị số
byte	1	8	Yes	-2 <sup>7</sup> to 2 <sup>7</sup> -1	-128 to 127
char	2	16	No	0 to 2 <sup>16</sup> -1	0 to 65535
short	2	16	Yes	-2 <sup>15</sup> to 2 <sup>15</sup> -1	-32768 to 32767
int	4	32	Yes	-2 <sup>31</sup> to 2 <sup>31</sup> -1	-2147483648 to 2147483647
long	8	64	Yes	-2 <sup>63</sup> to 2 <sup>63</sup> -1	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	32	Yes	-3.4028235E 38 to 3.4028235E38	
double	8	64	Yes	-1.7976931348623157E 308 to -1.7976931348623157E 308	
boolean	0	1	No		True false

- Khai báo biến các kiểu nguyên thủy và nhập các giá trị vào biến từ bàn phím thông qua Scanner như sau:

```

22 |   Scanner input = new Scanner(System.in);
23 |   byte byteVar = input.nextByte();
24 |   short shortVar = input.nextShort();
25 |   int intVar = input.nextInt();
26 |   long longVar = input.nextLong();
27 |   float floatVar = input.nextFloat();
28 |   double doubleVar = input.nextDouble();

```

### 2.8.3 Giá trị mặc định của biến có kiểu nguyên thủy

- Trong Java khi ta khai báo một biến kiểu nguyên thủy thì biến tự động được gán giá trị khởi tạo mà lập trình viên chưa cần phải khởi tạo nó. Sau đây là các giá trị khởi tạo ban đầu của các biến có kiểu tương ứng như sau:

Kiểu dữ liệu	Giá trị mặc định
byte	0
char	'\u0000'
short	0
int	0
long	0L
float	0.0f
double	0.0d
boolean	false

### 2.8.4 Các phép toán với các biến nguyên thủy kiểu số (Numeric).

Name	Meaning	Example	Result
+	Cộng		
-	Trừ		
*	Nhân		

/	Chia	1.0/2.0	0.5
%	Chia lấy phần dư	20%3	2
++	Tăng một đơn vị	x=5; x++	6
--	Giảm một đơn vị	X=5; x--	4

- Sau đây là ví dụ về chuyển đổi từ giây sang phút và giây: Nhập vào từ bàn phím một số có giá trị là số giây và quy đổi ra số phút và giây. Ví dụ: nhập vào 150 sẽ quy đổi ra 2 phút 30 giây.

```

36  public static void DisplayTime() {
37      Scanner input = new Scanner(System.in);
38      // Prompt the user for input
39      System.out.print("Enter an integer for seconds: ");
40      int seconds = input.nextInt();
41      int minutes = seconds / 60; // Find minutes in seconds
42      int remainingSeconds = seconds % 60; // Seconds remaining
43      System.out.println(seconds + " seconds is " + minutes
44          + " minutes and " + remainingSeconds + " seconds");
45  }

```

- Ở ví dụ này ta thấy “/” ở dòng 41 là phép chia. Tuy nhiên khi ta thực hiện phép chia này giữa các số nguyên với nhau thì kết quả sẽ là một số nguyên và không có phần dư sau dấu phẩy, thành ra khi thực hiện với số nguyên thì phép chia “/” lại trở thành phép chia lấy phần nguyên. Khi ta thực hiện phép chia “/” với số thực thì kết quả lại bình thường và ko phải là phép chia lấy phần nguyên.
- Cũng trong ví dụ trên “%” ở dòng 42 là phép chia lấy phần dư. Trong tính toán chúng ta chú ý điều này để nhận được kết quả chính xác.

```

run:
Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds
BUILD SUCCESSFUL (total time: 5 seconds)

```

- ✓ Như vậy là chúng ta đã hoàn thành nghiên cứu về các phép toán đổi với biến số nguyên thủy.

## 2.8.5 Chữ viết biểu thị giá trị trong Java (Literals)

- Chúng ta có thể sử dụng các ký tự để biểu thị giá trị kiểu nguyên thủy và ta gọi đó là các Literals. Trong thực tế viết code chúng ta sẽ sử dụng rất nhiều các literal để gán và khởi tạo giá trị cho các biến số. Chúng ta đi xét một ví dụ như dưới đây:

```

18     float x = 0.5f;
19     char unicode = '\u0041';
20     System.out.println("X = " + x);
21     System.out.println("unicode = " + unicode);
22
jb12.Welcome > main >
Output - JB12 (run) >
run:
X = 0.5
unicode = A
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Biến `X` và biến `unicode` được gán giá trị có sử dụng literal. Các literal này là các giá trị và không phải là biến vì vậy nó chỉ được dùng để biểu thị giá trị hoặc được gán vào các biến số. Dưới đây là liệt kê các literals chúng ta làm quen với chúng để sử dụng trong các tình huống thích hợp và không bị bỡ ngỡ khi gặp chúng.

Literals	Data type	Note
true	boolean	
false	boolean	
Tất cả các chữ cái, số, và ký tự VD: 'A', '1'	char	
'\u0041'	char	Biểu thị giá trị chữ 'A' tương ứng với mã Unicode  char unicode = '\u0041'
\n:	char	Ký tự đầu dòng
\r	char	Return (cuối dòng)
\t	char	Tab
\b	char	backspace

\f	char	form feed
\'	char	single quote
\"	char	double quote
\\\	char	backslash
43 053 0x2b 0xB 0X2b 0X2B	int	Các chữ số biểu thị giá trị 43 ở hệ số đếm thập phân, Bát phân, Thập lục phân.  Int x = 0x2b;
I L	long	Hậu tố biểu thị giá trị số kiểu long  long x = 43L;
f F	float	Hậu tố biểu thị giá trị kiểu số thực ví dụ:  float x = 5f
d D	double	Hậu tố biểu thị giá trị kiểu số thực ví dụ:  double x = 5d

- Đây là một ví dụ về việc sử dụng nhiều literal khác nhau để biểu thị cho giá trị số 43.

```

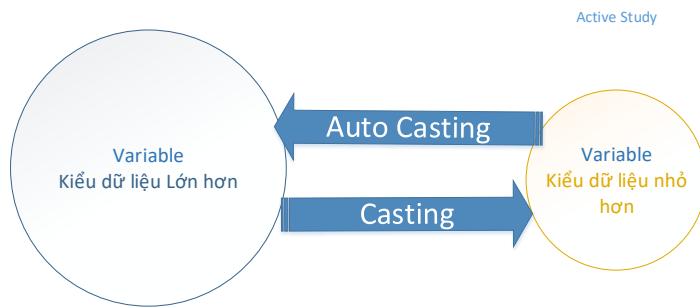
21     System.out.println("integer = " + 43);
22     System.out.println("integer = " + 053);
23     System.out.println("integer = " + 0x2b);
24     System.out.println("integer = " + 0x2B);
25     System.out.println("integer = " + 0X2b);
26     System.out.println("integer = " + 0X2B);
27
  jb12.Welcome > main >
Output - JB12 (run) X
  run:
  X = 0.5
  integer = 43
  integer = 43

```

- ✓ Như vậy là chúng ta đã đi tìm hiểu về Literals và cách sử dụng chúng, mong rằng các bạn không còn bỡ ngỡ khi bắt gặp chúng nữa.

#### 2.8.6 Chuyển đổi kiểu dữ liệu số trong Java (Casting)

- Trong Java chúng ta thường xuyên cần phải gán các giá trị từ biến số này sang biến số khác, các biến số có độ rộng khác nhau, vì vậy Java cung cấp cho chúng ta khả năng chuyển đổi kiểu dữ liệu rất linh hoạt. ta nhìn vào hình minh họa sau để thấy điều này



- Java tự động chuyển đổi (Auto Casting) một giá trị (chứa trong biến hoặc trong quá trình tính toán) có kiểu dữ liệu có Độ rộng nhỏ sang kiểu dữ liệu khác có độ rộng lớn hơn và bao phủ kiểu dữ liệu được chuyển đổi. VD: Một giá trị `int` có thể được chuyển sang kiểu `long`, `float`, `double`
- ví dụ 1 : `long x = 5;`  
 ví dụ 2 : `float y = x;`
- Xét ví dụ 1 Ta thấy số 5 là giá trị kiểu `int` được tự động chuyển sang kiểu `long` và gán sang biến `x` có kiểu `long` thay vì ta phải có phép gán chính xác như sau: `long x = 5L;`

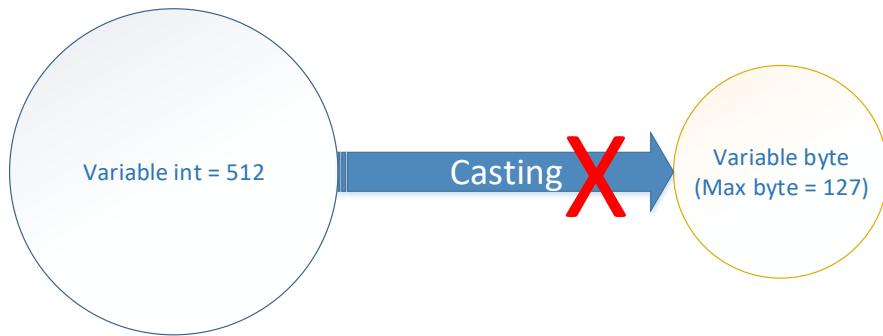
- o Xét ví dụ 2 ta thấy biến `x` có độ rộng nhỏ hơn biến `y`, hay nói cách khác biến `y` có kiểu `float` bao trùm kiểu `long` nên giá trị của biến `x` được tự động chuyển đổi sang kiểu `float` và gán vào biến `y`.
  
- Khi cần chuyển đổi một giá trị (Chứa trong biến hoặc trong quá trình tính toán) có kiểu dữ liệu có độ rộng lớn hơn sang kiểu dữ liệu có độ rộng nhỏ hơn thì cần phải chỉ đích danh kiểu dữ liệu cần chuyển đổi sang và ta còn gọi là ép kiểu (Casting). Nếu ta không chỉ đích danh thì trình biên dịch sẽ chỉ ra lỗi biên dịch. Tuy nhiên trong quá trình chuyển đổi kiểu dữ liệu này ta cần phải hoàn toàn chú ý kiểm soát được dữ liệu được chuyển đổi, vì nếu giá trị đem đi gán có thể sẽ lớn hơn độ rộng của biến đích thì sẽ sai logic và giá trị nhận được sẽ có thể là Infinity.

Ví dụ 1: `int a = (int)1.7;`

Ví dụ 2: `int a = 5; char y = (char)a;`

Ví dụ 3: `int b = 512; byte z = (byte)b;`

Active Study



- Kết quả tính toán sẽ được tự động chuyển đổi sang kiểu dữ liệu lớn hơn của một trong hai toán hạng

Ví dụ : `float x = 0; x = (float)5/4; x = 5/(float)4;`

Xét ví dụ trên thì kết quả tính toán của 2 cách trên trên là như nhau và được chuyển đổi về kiểu dữ liệu float thay vì kiểu int.

- Xét ví dụ :

`float x = 5.5; int y = (int)x`

Chú ý kết quả sau khi thực hiện thì `y` sẽ mang giá trị 5 còn `x` vẫn sẽ mang giá trị 5.5 bởi vì `y` là một kiểu số nguyên.

- Hai trường hợp như ví dụ sau đây là tương đương nhau

Ví dụ 4 : `float x = 1.7f; int sum = 0; sum += x;`

Ví dụ 5 : `float x = 1.7f; int sum = 0; sum = (int)(sum + x)`

- Các toán hạng kiểu `byte` hay `short` sẽ được chuyển sang kiểu `int` trước khi thực hiện phép toán.
- Trong Java không thể chuyển biến kiểu `int` và kiểu `boolean` như trong ngôn ngữ C/C++ được.

✓ Vậy là chúng ta đã nghiên cứu xong về chuyển đổi kiểu dữ liệu số trong Java.

## 2.9 Kiểu dữ liệu char

- Kiểu `char` là kiểu dữ liệu đặc biệt có độ rộng 2 byte và tương ứng với miền giá trị từ 0 đến  $2^{16}-1$  (0 - 65535)
- Java là ngôn ngữ lập trình hỗ trợ tốt Unicode. Các biến kiểu dữ liệu `char` có giá trị là các mã `Unicode 16 bit` của các ký tự được quy định trong bảng mã `Unicode`. Ta xem xét bảng sau để thấy được một phần của bảng mã `Unicode` này:

Ký tự	Giá trị mã ký tự cơ sở 10	Giá trị mã ký tự Unicode
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

- Bộ mã Unicode hoàn toàn tương thích với bộ mã `ASCII 8 bit`. Và `Unicode 16 bit` chính là phần mở rộng của bảng mã này.
- Để biểu thị toàn bộ chữ viết trên toàn thế giới thì bộ mã Unicode 16 bit không thể đáp ứng được, vì vậy Java hỗ trợ bộ mã Unicode bổ sung gọi là “supplementary characters” lên tới 1,112,064 characters thay vì 65535 như kiểu `char`.
- Kiểu dữ liệu `char` cũng được tính toán và chuyển đổi (Casting) như một kiểu dữ liệu số,

```

22 |     char ch = 64;
23 |     ch = (char)(ch + 1);
24 |     System.out.println(ch);
25 |
Output - JavaCoreBasic (run) ×
run:
A
  
```

- khi được xử lý dưới dạng ký tự ví dụ như: hiển thị ra màn hình hoặc cộng chuỗi thì một giá trị kiểu char sẽ được xử lý đúng như là một ký tự.

```

27 |     char ch = 65;
28 |     String welcome = "XIN CH" + ch + "O BAN";
      |     System.out.println(welcome);
  
```

Output - JavaCoreBasic (run) ×

```

run:
XIN CHAO BAN
  
```

- Các literals đặc biệt và giá trị của kiểu char.

Escape sequence	Name	Unicode code	Decimal Value
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Cariage Return	\u000D	13
\\\	Backslash	\u005C	92
\\"	Double Quote	\u0022	34

ký tự “\” là ký tự escape character để hiển thị các ký tự đặc biệt như bảng trên.

VD: System.out.println("\t is a tab character");

Sẽ hiển thị \t một tab ở trước như sau: “ is a tab character”

- Hai biến hoặc số kiểu char có thể được so sánh và sử dụng các phép so sánh “<, <=, >, >=, ==,” như với kiểu số

```

12     public class CharExample {
13         public static void main(String[] args) {
14             char a = 65;
15             char b = 66;
16             if(a > b){
17                 System.out.println("Kết Quả: " + a + " > " + b);
18             }else{
19                 System.out.println("Kết Quả: " + a + " < " + b);
20             }
21         }
22     }

```

Kết quả sau khi chạy như sau:

Output - JavaCoreBasic (run) X

```

run:
Kết Quả: A < B

```

Từ ví dụ trên ta thấy hai biến số a và b kiểu char được so sánh với nhau tại dòng 16 như là 2 số nguyên và khi in kết quả ra màn hình thì hai ký tự A và B tương ứng với mã 65 và 66 được cộng vào chuỗi ký tự và in ra màn hình.

- ✓ Như vậy là chúng ta đã hoàn thành nghiên cứu về kiểu dữ liệu char.

## 2.10 Toán tử trong Java

### 2.10.1 Toán tử số học

- Các toán tử này áp dụng cho các số hoặc biến số trong quá trình thực hiện giải thuật tính toán.

Name	Meaning	Example x	Result x
+	Cộng	int x = 5; x = x + 1;	6
-	Trừ	int x = 5; x = x - 1;	4
*	Nhân	int x = 5; x = x*6;	30
/	Chia	float x = 1.0/2;  int x = 12/10;	0.5  1
%	Chia lấy phần dư	int x = 20%3;	2

++	Tăng một đơn vị	int x = 5; x++;	6
--	Giảm một đơn vị	Int x = 5; x--;	4
+=	Công giá trị bên phải vào biến bên trái và gán giá trị tổng cho biến bên trái	int x=5; x += 1;	6
-=	Biến bên trái trừ giá trị bên phải và gán giá trị hiệu cho biến bên trái	int x = 5; x -=1;	4
*=	Nhân giá trị bên phải vào biến bên trái và gán giá trị tích cho biến bên trái	int x =20; x *=4;	80
/=	Biến bên trái chia giá trị bên phải và gán giá trị thương cho biến bên trái	int x =20; x /=4;	5
%=	Biến bên trái chia giá trị bên phải và gán giá trị phần dư cho biến bên trái	int x = 20; x %=3;	2

- ✓ Vậy là chúng ta đã nghiên cứu xong về toán tử số học trong Java.

### 2.10.2 Toán tử logic

- Ta xét một ví dụ như sau:
  - o So sánh hai số nguyên xem số nào lớn hơn

```

23   int a = 10;
24   int b = 15;
25   if(a > b){
26       System.out.println("a > b");
27   } else if(a < b){
28       System.out.println("a < b");
29   }else{
30       System.out.println("a = b");
31   }

```

- o Chúng ta thấy **a** và **b** được so sánh với nhau, ta gọi (**a > b**) là biểu thức logic và kết quả trả ra là một giá trị **boolean** mang giá trị **true** hoặc **false**
- o Vậy các toán tử logic sẽ đều trả ra các giá trị **true** hoặc **false** mà thôi.
- Trong lập trình các mệnh đề logic hay còn gọi là biểu thức logic là rất phổ biến vì nó đóng vai trò quan trọng trong các giải thuật trong các chương trình phần mềm nói chung.
- Các toán tử logic thường được sử dụng trong các biểu thức logic và kết hợp với các cấu trúc điều khiển **if**, **while**, **for** để điều khiển các cấu trúc này theo một trình tự logic mà chúng ta mong muốn.
- Hầu hết các toán tử logic đều đã quen thuộc với chúng ta từ trong toán học rồi. Chúng ta xét bảng dưới sau để làm quen lại với các toán tử logic này.

Tên Toán tử	Ý nghĩa	Ví dụ <code>int x= 4; boolean y;</code>	Giá trị y sau ví dụ
<b>==</b>	So sánh bằng	<code>y = (x == 4);</code>	<code>true</code>
<b>!=</b>	So sánh khác	<code>y = (x != 4);</code>	<code>false</code>
<b>&gt;</b>	So sánh lớn hơn	<code>y = ( x &gt; 5);</code>	<code>false</code>
<b>&lt;</b>	So sánh nhỏ hơn	<code>y = (x &lt; 5);</code>	<code>true</code>
<b>&gt;=</b>	So sánh lớn hơn hay bằng	<code>y = (x &gt;= 5);</code>	<code>false</code>
<b>&lt;=</b>	So sánh nhỏ hơn hay bằng	<code>y = (x &lt;= 5);</code>	<code>true</code>
<b>  </b>	OR (biểu thức logic)	<code>y = ((x &gt; 5)    (x &lt; 10));</code>	<code>true</code>

<b>&amp;&amp;</b>	AND (biểu thức logic)	<code>y = (x &gt; 5) &amp;&amp; (x &lt; 10);</code>	false
<b>&amp;</b>			
<b>!</b>	NOT (biểu thức logic)	<code>y = !true;</code>	false

- Trong bảng trên ta thấy sự xuất hiện của toán tử | và || cũng như & và &&. Về chức năng thì chúng tương đồng nhau nhưng cách thức thực hiện thì lại có sự khác nhau. Cụ thể như sau:

Xét ví dụ sau:

```

15  int x = 3;
16  int y = 10;
17  if ((x > 2) | (y++ < 20)) {
18      System.out.println("True");
19  }
20  System.out.println("x = " + x);
21  System.out.println("y = " + y);
22 }
```

Kết quả :

Output - JavaCoreBasic (run) ×

run:	True
x =	3
y =	11

Cũng đoạn code trên ta thay toán tử “|” bằng toán tử “||” thi kết quả thực hiện lại có sự khác biệt:

```

15  int x = 3;
16  int y = 10;
17  if ((x > 2) || (y++ < 20)) {
18      System.out.println("True");
19  }
20  System.out.println("x = " + x);
21  System.out.println("y = " + y);
22 }
```

Kết quả :

Output - JavaCoreBasic (run) ×

run:	True
x =	3
y =	10

- o Đối với toán tử logic “|” thì chúng sẽ thực hiện cả hai biểu thức logic trong dấu ngoặc tại dòng 17 và sau đó đưa ra kết quả. Biểu thức logic thứ 1 (`x>2`) trả ra giá trị true. Biểu thức logic thứ 2 (`y++ < 20`) cũng được thực hiện. Vậy thế là `y` được cộng lên 1 giá trị trong biểu thức so sánh logic thứ 2 này. Và kết quả “**Hoặc**” của hai biểu thức logic là `true`;
- o Đối với toán tử logic “||” thì tại dòng 17 chúng sẽ thực hiện biểu thức logic thứ nhất (`x >2`), do thỏa mãn điều kiện true nên không cần thực hiện biểu thức logic thứ 2 (`y++ < 20`) nữa vậy nên `y` sẽ không được cộng thêm 1 giá trị trong biểu thức thứ 2.

**Kết luận:** Ta nên ưu tiên sử dụng “`||`” thay vì “`&`” vì sự tối ưu của chúng trong cách thực hiện và chú ý đến logic của nó.

- Tương tự như vậy đối với phép “`&`” và “`&&`”

```

15 |     int x = 3;
16 |     int y = 10;
17 |     if ((x < 2) & (y++ < 20)) {
18 |         System.out.println("True");
19 |     }
20 |     System.out.println("x = " + x);
21 |     System.out.println("y = " + y);

```

Kết quả:

Output - JavaCoreBasic (run) X

run:  
x = 3|  
y = 11

Thay phép “`&`” bằng phép “`&&`”

```

15 |     int x = 3;
16 |     int y = 10;
17 |     if ((x < 2) && (y++ < 20)) {
18 |         System.out.println("True");
19 |     }
20 |     System.out.println("x = " + x);
21 |     System.out.println("y = " + y);

```

Kết quả:

Output - JavaCoreBasic (run) X

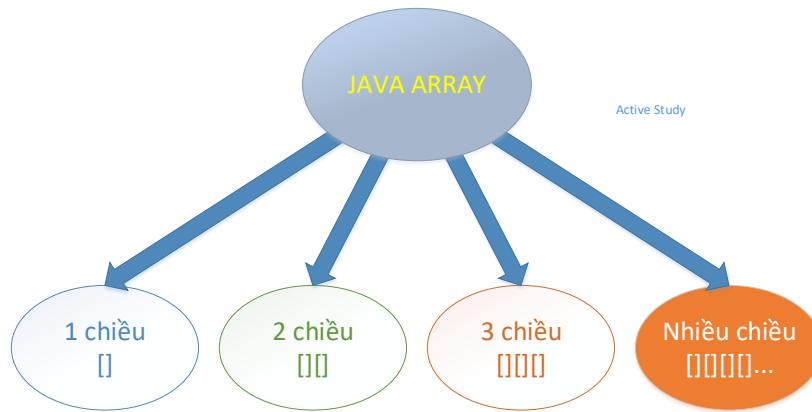
run:  
x = 3|  
y = 10

**Kết luận tương tự:** Ta nên ưu tiên sử dụng “`&&`” thay vì “`&`” vì sự tối ưu của chúng trong cách thực hiện và chú ý đến logic của nó.

- ✓ Vậy là chúng ta đã nghiên cứu xong về toán tử logic trong Java.

## 2.11 Kiểu dữ liệu mảng trong Java

- Mảng là một kiểu dữ liệu phổ biến trong tất cả các ngôn ngữ lập trình. Kiểu dữ liệu mảng dùng để chứa tập hợp các dữ liệu giống nhau trong quá trình xử lý dữ liệu. Mảng đóng vai trò quan trọng trong quá trình nghiên cứu Java core basic và trong quá trình rèn luyện tư duy giải thuật của chúng ta.
- Tuy nhiên trong Java core advance thì các kiểu dữ liệu collection và map sẽ được sử dụng nhiều hơn và thay thế kiểu mảng bởi tính linh động và tính chuyên dụng của chúng, chúng ta sẽ tìm hiểu các kiểu dữ liệu mở rộng này ở phần Java core advance.
- Trong Java hỗ trợ mảng một chiều, mảng 2 chiều, mảng 3 chiều. Oh thật kinh ngạc thực ra là mảng n chiều.



- Và quả thực là ta có thể khai báo được một mảng 10 chiều và thậm chí nhiều hơn thế nữa.

```

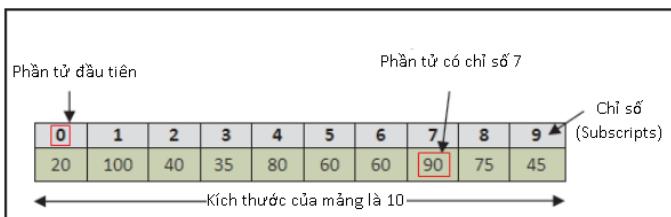
15 |   |   | //Khai bao mot mang n chieu
16 |   |   | int arrInt[][][][][][][][][][] = new int[1][2][3][4][5][6][7][8][9][10];
  
```

- Phần tiếp theo sau đây chúng ta sẽ đi tìm hiểu về mảng 1 chiều và mảng 2 chiều.

### 2.11.1 Mảng một chiều trong Java

#### 2.11.1.1 Khai báo và khởi tạo mảng một chiều

- 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.



- Sau đây là 2 kiểu khai báo một biến mảng
  - Kiểu 1      <kiểu dữ liệu> <tên mảng>[];
  - Kiểu 2      <kiểu dữ liệu>[] <tên mảng>;

➔ Hai cách khai báo này là như nhau và chúng ta dùng cách nào cũng được.

- Ta đi xét ví dụ sau:

```

15 |     //Khai báo một mảng
16 |     int arrInt[];
17 |     //hoặc
18 |     int[] arrInt1;
19 |     //hoặc
20 |     int[] arrInt2, arrInt3;

```

- Trước khi sử dụng mảng thì ta cần bắt buộc phải khởi tạo số phần tử cho mảng bằng từ khóa **new**.  
Ta đi xét ví dụ sau về khởi tạo phần tử mảng trong Java:

```

15 |     int arrInt[];
16 |     arrInt = new int[100];

```

Hoặc chúng ta có thể khai báo và khởi tạo trên cùng một dòng như sau đều được:

```

18 |     int arrInt4[] = new int[100];

```

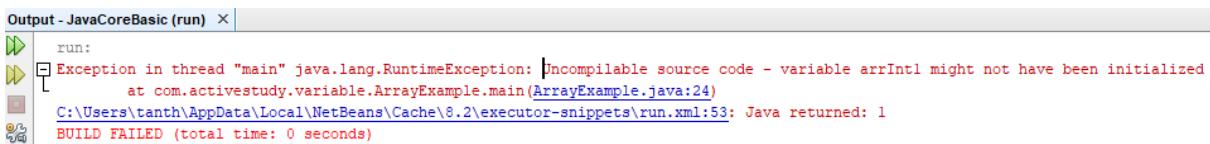
- Chúng ta có thể cấp phát động số lượng phần tử của mảng thông qua biến số như sau

```

47 |     int n = 5;
48 |     int arrInt[] = new int[n];
49 |

```

- Việc cấp phát động như trên có thể giúp sử dụng số phần tử mảng một cách linh động, đúng đủ số lượng và tránh lãng phí bộ nhớ khi cấp phát tĩnh ngay từ đầu.
- Chúng ta phải nhớ cấp phát bộ nhớ cho mảng trước khi sử dụng. Lỗi sử dụng mảng mà chưa đi khởi tạo các phần tử mảng rất hay gặp phải và các bạn sẽ nhận về lỗi biên dịch vì chưa khởi tạo các phần tử mảng



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

```

46 } // ...
47 int arrInt[] = {1, 2, 3};
48 char arrChar[] = {'a', 'b', 'c'};
49 String arrStrng[] = {"ABC", "EFG", "GHI"};
50

```

### 2.11.1.2 Truy xuất đến các phần tử mảng 1 chiều

- Các phần tử của mảng được truy xuất thông qua chỉ số của nó, chỉ số này đặt giữa cặp dấu ngoặc vuông ([]). 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.

Ví dụ về truy xuất tới phần tử mảng như sau:

```

47 int arrInt[] = {1, 2, 3};
48 int x = arrInt[0]; // x sẽ có giá trị là 1.
49 int y = arrInt[1]; // y sẽ có giá trị là 2.
50 int z = arrInt[2]; // z sẽ có giá trị là 3.
51

```

Ví dụ khác về truy cập tới phần tử mảng:

```

18 int arrInt1[] = {1,2,3,4,5};
19 for(int i = 0; i < arrInt1.length; i++){
20     System.out.print(arrInt1[i]);
21 }

```

Kết quả trả về như sau:

Output - JavaCoreBasic (run) ×

run:  
12345

- Trong quá trình truy cập tới các phần tử mảng bằng chỉ số trong dấu [index] chúng ta hay gặp phải lỗi khi chỉ số index vượt quá số lượng phần tử mảng và nhận về lỗi runtime như ví dụ sau:

```

23 int arrInt4[] = new int[100];
24 System.out.println("Truy cap toi phan tu mang vuot qua index: " + arrInt4[101]);

```

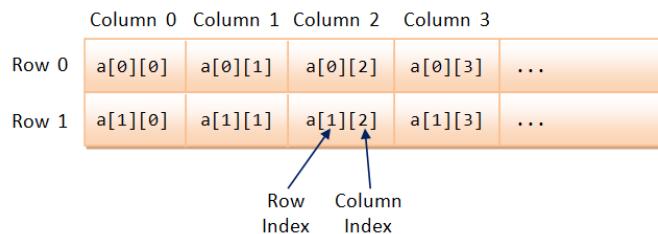
Kết quả nhận được khi chạy như sau:

```
Output - JavaCoreBasic (run) ×
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 101
at com.activestudy.variable.ArrayExample.main(ArrayExample.java:24)
```

## 2.11.2 Mảng hai chiều trong Java

### 2.11.2.1 Khai báo và khởi tạo mảng hai chiều

- Vừa rồi chúng ta tìm hiểu mảng 1 chiều, Mảng hai chiều cũng giống như mảng 1 chiều đó là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu, nhưng điểm khác biệt là nó hai chiều như không gian 2 chiều đó là có hàng và cột, và mỗi phần tử trong mảng được truy xuất thông qua 2 chỉ số hàng và cột.
- Để hình dung về mảng 2 chiều ta hình dung nó như hình bên dưới biểu diễn một mảng 2 chiều a có 2 hàng và 4 cột. Các giá trị được sắp vào các vị trí hàng cột như một ma trận (Ma trận trong toán học cao cấp)



- Mảng hai chiều cũng được khai báo theo 2 kiểu như sau:

- Kiểu 1

`<kiểu dữ liệu> <tên mảng>[][];`

- Kiểu 2

`<kiểu dữ liệu>[][] <tên mảng>;`

Hai cách khai báo này không khác nhau về ý nghĩa chỉ khác ở điểm [] nằm ở sau kiểu dữ liệu hay sau tên biến mà thôi.

Chúng ta xét ví dụ sau:

```

47 |     int x = 5;
48 |     int[][] arr = new int[x][x];
49 |     arr[0][0] = 1;
50 |     arr[0][1] = 2;

```

### 2.11.2.2 Truy xuất đến các phần tử mảng 2 chiều

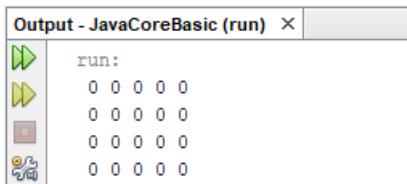
- Việc truy cập đến các phần tử mảng hai chiều với 2 chỉ số dòng và cột của hai chiều tương ứng. Ta xét một ví dụ sau về việc truy cập tới từng phần tử của mảng 2 chiều:

```

24 |     int n = 4;
25 |     int m = 5;
26 |     int arrInt5[][] = new int[n][m];
27 |     for(int i = 0; i <n; i ++){
28 |         for(int j = 0; j < m; j++ ){
29 |             System.out.print(" " + arrInt5[i][j]);
30 |         }
31 |         System.out.println("");
32 |

```

Kết quả sau khi chạy sẽ như sau:



- ✓ Vậy là chúng ta đã hoàn thành nghiên cứu về mảng, mảng 1 chiều và mảng hai chiều cùng với cách truy cập tới các phần tử mảng.

## 2.12 Cấu trúc điều khiển trong Java

- Cấu trúc điều khiển trong Java khá giống với C, C++ và các ngôn ngữ khác bao gồm cấu trúc điều khiển rẽ nhánh if...else, switch... case và cấu trúc lặp while, do...while, for. Các cấu trúc này vô cùng quan trọng, khi sử dụng đúng và khéo léo thì chương trình của chúng ta sẽ vô cùng mềm dẻo. các cấu trúc này là cơ sở để giúp chúng ta xây dựng các giải thuật trong chương trình.
- Cấu trúc rẽ nhánh cho phép điều khiển luồng thực thi hoặc bỏ qua một khối lệnh dựa vào giá trị true hoặc false của mệnh đề điều kiện. Đó là hai cấu trúc if và switch.

- Cấu trúc lặp cho phép thực thi lặp lại khối lệnh liên tục với số lần lặp biết trước hoặc không biết trước phụ thuộc vào điều kiện true hay false của mệnh đề điều kiện lặp. Đó là cấu trúc while và for.

Các phần sau đây chúng ta sẽ đi tìm hiểu về từng cấu trúc này.

### 2.12.1 Cấu trúc điều kiện rẽ nhánh if...else trong Java

- Cấu trúc điều kiện rẽ nhánh if có 3 hình mẫu như sau:

Kiểu 1:

```

14 |     boolean condition1 =  true;
15 |     if(condition1){
16 |         //Khối lệnh 1
17 |     }

```

- Nếu điều kiện condition1 có giá trị true thì khối lệnh 1 sẽ được thực thi, ngược lại condition1 có giá trị false thì khối lệnh 1 sẽ bị bỏ qua và không thực thi khi chạy.

Kiểu 2:

```

14 |     boolean condition1 =  true;
15 |     if(condition1){
16 |         //Khối lệnh 1
17 |     }else{
18 |         //Khối lệnh 2
19 |     }

```

- Nếu điều kiện condition1 có giá trị true thì khối lệnh 1 sẽ được thực thi và bỏ qua khối lệnh 2, ngược lại condition1 có giá trị false thì khối lệnh 2 sẽ được thực thi và bỏ qua khối lệnh 1 khi chạy.

Kiểu 3:

```

14 |     boolean condition1 =  true, condition2=true, conditionN=false;
15 |     if(condition1){
16 |         //Khối lệnh 1
17 |     }else  if(condition2){
18 |         //Khối lệnh 2
19 |     }else  if(conditionN){
20 |         //Khối lệnh n
21 |     }else{
22 |         //Khối lệnh n+1
23 |     }

```

- Nếu điều kiện `condition1` có giá trị `true` thì khối lệnh 1 sẽ được thực thi và bỏ qua các khối lệnh phía dưới, ngược lại `condition1` có giá trị `false` thì các `condition` sau sẽ được xem xét và thực thi các khối lệnh nào có `condition` tương ứng là `true` và bỏ qua các điều kiện và khối lệnh sau nó.
- Nếu không có `condition` nào trả về giá trị `true` thì sẽ chỉ thực thi khối lệnh cuối cùng trong `else` nếu nó.

Chúng ta xem xét một ví dụ minh họa như sau:

Xem xét bài toán giải phương trình bậc 2:

```

25      int a = 5,b = 10,c = 4;
26      int delta = b*b-4*a*c;
27      System.out.println("delta = " + delta);
28      if (delta > 0) {
29          System.out.println("Phuong trinh co hai Nghiem phan biet");
30      } else if (delta == 0) {
31          System.out.println("Phuong trinh co nghiem kep");
32      } else {
33          System.out.println("Phuong trinh vo nghiem");
34      }

```

Kết quả chạy chương trình như sau:

Output - JavaCoreBasic (run) #2 ×

run:  
delta = 20  
Phuong trinh co hai Nghiem phan biet

Theo như kết quả thì dòng 29 được thực thi do `delta` = 20 thỏa mãn điều kiện `delta > 0` và các lệnh từ dòng 30 tới dòng 33 đã bị bỏ qua.

- ✓ Vậy là chúng ta đã tìm hiểu xong về cấu trúc điều khiển rẽ nhánh if.

### 2.12.2 Cấu trúc điều kiện rẽ nhánh switch-case trong Java

- Cấu trúc `if` cho phép lựa chọn chỉ 1 khối lệnh được phép thực hiện hoặc không khôi nào cả. Cấu trúc `switch` cho phép lựa chọn một hoặc nhiều hoặc không khôi lệnh nào chạy cả. Chúng ta xét ví dụ sau:

```

30     int x = 3;
31     switch (x) {
32         case 1:
33             System.out.println("The value of x is 1.");
34             break;
35         case 2:
36             System.out.println("The value of x is 7.");
37             break;
38         case 3:
39             System.out.println("The value of x is 3.");
40         case 4:
41             System.out.println("The value of x is 4.");
42         case 5:
43             System.out.println("The value of x is 5.");
44     default:
45         System.out.println("The value of x is default.");
46     }
47

```

jb12.Welcome > main > x >

Output - JB12 (run) >

```

run:
The value of x is 3.
The value of x is 4.
The value of x is 5.
The value of x is default.
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Nếu  $x = 1$  thì chỉ có khối lệnh 1 được thực hiện. kết quả in ra sẽ như sau:

```

run:
The value of x is 1.
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Nếu  $x= 3$  thì 3 khối lệnh được thực hiện.

```

run:
The value of x is 3.
The value of x is 4.
The value of x is 5.
The value of x is default.

```

- Nguyên nhân có sự khác biệt này là do lệnh `break` xuất hiện ở khối lệnh 1. Khi thực hiện lệnh `break` sẽ bỏ qua tất cả các lệnh sau đó của khối lệnh `switch`. Chúng ta sẽ bàn tới lệnh `break` này ở cấu trúc lặp tiếp theo nữa.
- Nếu  $x = 6$  thì chúng ta thấy sẽ không trùng với giá trị nào được chỉ ra trong `case`. Lúc này khối lệnh ở `default` sẽ được thực thi. Nếu không có khối lệnh `default` này thì sẽ không có khối lệnh nào được thực thi cả.
- Kiểu giá trị mà biểu thức trong ngoặc của `switch(x)` nhận bao gồm những kiểu sau: `byte`, `short`, `int`, `char`, `String`, `enum` và không cho phép các kiểu dữ liệu sau: `long`, `float`, `double`, `boolean`.
- Từ khóa và khối lệnh `default` có thể có hoặc không.
- Trong một biểu thức `switch` không cho phép trùng lặp các giá trị ở các `case`.

- Biểu thức trong **switch** không cho phép khai báo một biến nhưng cho phép một biểu thức tính toán.

Ví dụ sau là không được phép : **switch (int x)**

Ví dụ sau là được phép **switch (x + y)** hoặc **switch (x++)**

- Chú ý kiểu hoặc độ lớn của giá trị tại **case** phải tương xứng với kiểu và độ rộng của biến số trong biểu thức **switch()**

Ví dụ

```

13  public static void main(String[] args) {
14      byte x = 3;
15      switch(x) {
16          case 3:
17              System.out.println("x = 3");
18              break;
19          case 200:
20              //Không phù hợp với kiểu byte;
21              // giá trị lớn nhất của byte = 127
22              break;
23      }

```

- Lệnh **switch** sẽ thực thi nhanh hơn nếu giá trị của các **case** được sắp xếp theo thứ tự tăng dần.
- ✓ Vậy là chúng ta đã tìm hiểu xong về cấu trúc điều khiển rẽ nhánh **switch**.

### 2.12.3 Cấu trúc Lặp While và do-while trong Java

- Java cung cấp 2 cấu trúc lặp **while**, nhiệm vụ thì giống nhau nhưng có một chút khác biệt trong thứ tự thực thi khối lệnh trong **while** và kiểm tra điều kiện **while**.
- Quan sát vào hình minh họa dưới đây để thấy được cách hoạt động và sự khác biệt giữa 2 cấu trúc **while**.



Sau đây ta sẽ xét hai ví dụ về cấu trúc này để thấy được điều này được điểm khác biệt như sau:

```

15 |   int i = 0;
16 |   while (i < 3) {
17 |     System.out.println("Vong lap thu " + i);
18 |     i++;
19 |

```

- Ban đầu `i = 0` thỏa mãn điều kiện `while : i < 3` vì vậy khối lệnh trong `while` được thực thi cho tới khi `i` được tăng lên đến `3` thì dừng lại. như vậy biến `i` được điều khiển theo logic tăng dần để đạt điều kiện lặp 3 lần và dừng lại.

Giá trị đầu ra sẽ như sau:

```

Output - JavaCoreBasic (run) #2 ×
run:
Vong lap thu 0
Vong lap thu 1
Vong lap thu 2

```

Chúng ta đi xét tiếp ví dụ về cấu trúc do while sau:

```

21 |   int i = 4;
22 |   do {
23 |     System.out.println("Vong lap thu " + i);
24 |     i++;
25 |   } while (i < 3);

```

Giá trị đầu ra sẽ như sau:

```

Output - JavaCoreBasic (run) #2 ×
run:
Vong lap thu 4

```

- Tại dòng 21 ban đầu  $i = 4$  vậy là đã không thỏa mãn điều kiện lặp while tại dòng 25. Nhưng khi chương trình vẫn chạy qua dòng 23 và 23 trước sau đó mới đến dòng 25, sau đó do điều kiện  $i < 3$  không thỏa mãn nên vòng lặp do while bị dừng lại và chương trình sẽ chạy tiếp quá.
  - Như vậy chúng ta thấy khôi lệnh trong **do while** sẽ được thực thi ít nhất 1 lần và đây là sự khác biệt giữa 2 cấu trúc **while** và **do...while**. Có thể ví **do..while** là “Trảm trước tâu sau” còn **while** là “tâu trước trảm sau” ☺☺☺
- ✓ Vậy là chúng ta đã tìm hiểu xong về cấu trúc điều khiển lặp **while** và **do while**.

#### 2.12.4 Cấu trúc lặp for... trong Java

##### 2.12.4.1 Cú pháp và nguyên lý hoạt động của vòng lặp For

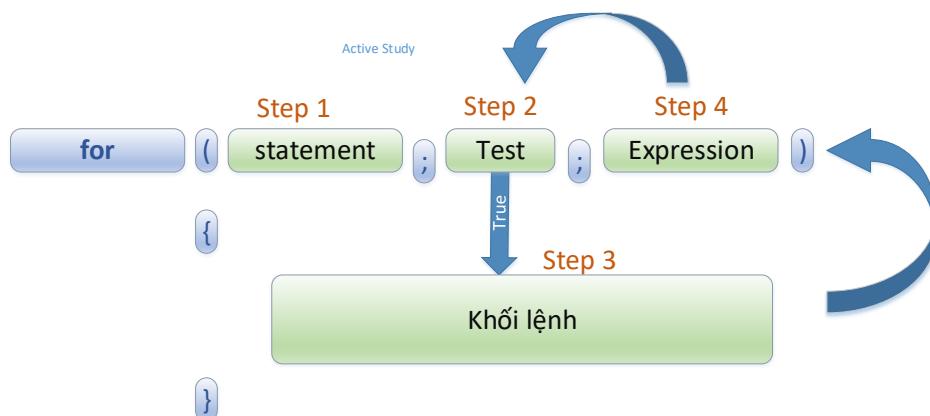
- Java cung cấp cấu trúc lặp **for** giúp chúng ta thực hiện khôi lệnh với số lần lặp lại hữu hạn. Nếu đã xác định trước số lần lặp ta nên sử dụng cấu trúc for để chương trình của chúng ta được trong sáng dễ hiểu và an toàn hơn.
- Ví dụ về cấu trúc for như sau:

```

47 |   for (int i = 0; i < 3; i++) {
48 |     System.out.println("The value of i: " + i);
49 |

```

Xét cấu trúc sau:



Trong đó

- <statement>:
  - o Do được chạy đúng một lần đầu khi thực thi bắt đầu vòng for nên thường được sử dụng để khai báo và khởi tạo giá trị của biến để điều khiển vòng lặp for.
- <test> :
  - o là biểu thức trả ra giá trị **true** hoặc **false**. Nếu trả ra giá trị **true** thì khối lệnh trong **for** sẽ được thực hiện và lặp lại cho tới khi biểu thức <test> này trả ra giá trị **false**. Mỗi một lần lặp biểu thức này lại được thực thi và kiểm tra giá trị trả về.
- <expression>:
  - o là biểu thức được thực thi sau mỗi khi thực thi xong khối lệnh trong **for**. Biểu thức thường được sử dụng như là biểu thức điều khiển vòng lặp for.

➔ Ba biểu thức này có thể để trống nhưng vẫn phải có dấu ; để ngăn cách với hai biểu thức còn lại.

- Chúng ta đi xét ví dụ dưới đây:

```

47 |     for (int i = 0; i < 3; i++) {
48 |         System.out.println("The value of i: " + i);
49 |

```

- o <Statement> tương ứng là: **int i = 0;**
- o <Test> tương ứng là **i < 3;**
- o <Expression> tương ứng là **i++**
- o Giá trị biến khởi tạo **i = 0**, điều kiện dừng là **i < 3** và mỗi vòng lặp i được tăng lên 1 giá trị bằng biểu thức **i++**. Kết quả thực hiện đoạn lệnh như sau:

```

run:
The value of i: 0
The value of i: 1
The value of i: 2
BUILD SUCCESSFUL (total time: 0 seconds)

```

- ✓ Vậy là chúng ta đã hoàn thành nghiên cứu về Cú pháp và nguyên lý hoạt động của vòng lặp For

### 2.12.4.2 Vòng lặp for each

- Trong Java cung cấp các cấu trúc dữ liệu dạng tập hợp đó là **Array** hoặc **collection**. Để duyệt tự động từng phần tử của các cấu trúc dữ liệu này, java cung cấp cấu trúc lặp **for-each**. Và cấu trúc như sau:

```

for (<variable> : <array hoặc collection>) {
    // the block code
}

```

- Chúng ta sẽ xem xét cấu trúc này sau khi tìm hiểu về các cấu trúc dữ liệu **Array** và **collection**. Tuy nhiên chúng ta có thể xem trước thông qua ví dụ sau:

```

47 |     int[] myArray = new int[3];
48 |     myArray[0]= 10;
49 |     myArray[1] = 20;
50 |     myArray[2] = 30;
51 |     for(int i : myArray) {
52 |         System.out.println (i );
53 |

```

Kết quả đầu ra sẽ như sau:

```

run:
10
20
30
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Chúng ta nên ưu tiên sử dụng for each để tránh các lỗi về truy cập tới các phần tử arr hoặc collection không tồn tại như đã chỉ ra ở phần truy xuất tới phần tử mảng ở phần trước.
- ✓ Vậy là qua phần này ta đã biết cách sử dụng vòng lặp for each.

### 2.12.4.3 Các phương bỏ qua và thoát ra khỏi vòng lặp

- Để làm chủ hoàn toàn các vòng lặp thì ta phải nắm được các phương pháp dừng hoặc bỏ qua vòng lặp như sau:
- Cách 1: Phương pháp thông thường.
  - o Để dừng vòng lặp for, while theo cách thông thường thì ta dừng theo đúng logic bằng điều kiện **<test>** trả ra giá trị false.

- Cách 2: Dừng vòng lặp bằng từ khóa **break**;
  - o Trong khôi lệnh của cấu trúc lặp for hoặc while khi gặp lệnh break; thì vòng lặp sẽ ngay lập tức bỏ qua các lệnh tiếp theo trong khôi lệnh và dừng lại bất kể điều kiện trong biểu thức <test> có thể nào vì nó sẽ không được gọi tới nữa, chương trình sẽ tiếp tục thực hiện các lệnh sau vòng lặp. chúng ta xét ví dụ sau:

```

47 |         for (;;) {
48 |             System.out.println("we not for ever.");
49 |             break;
50 |         }
51 |         System.out.println("we continue...");
52 |

```

Kết quả đầu ra sau khi thực hiện sẽ như sau:

```

run:
we not for ever.
we continue...
BUILD SUCCESSFUL (total time: 0 seconds)

```

- o Chú ý **break** chỉ có thể sử dụng bên trong một cấu trúc lặp hoặc **switch**.
- Cách 3: Bỏ qua vòng lặp hiện tại bằng từ khóa **continue**;
  - o Biểu thức **continue** xuất hiện trong các cấu trúc lặp như **while** và **for** có tác dụng bỏ qua tất cả các câu lệnh ở ngay đằng sau nó của vòng lặp hiện tại và tiếp tục thực hiện vòng lặp kế tiếp.
  - o Xét ví dụ sau:

```

47 |         for (int i = 0; i < 5; i++) {
48 |             if (i == 3) {
49 |                 continue;
50 |             }
51 |             System.out.println("The value of i is " + i);
52 |

```

Kết quả đầu ra sẽ như sau:

```

run:
The value of i is 0
The value of i is 1
The value of i is 2
The value of i is 4
BUILD SUCCESSFUL (total time: 0 seconds)

```

Khi *i* đạt giá trị 3 thì câu lệnh hiển thị `System.println ("The value of i is " + i);`; đằng sau nó đã bị bỏ qua không thực hiện và chuyển ngay qua vòng lặp kế tiếp.

- o Chú ý `continue` chỉ có thể được sử dụng bên trong một cấu trúc lặp.

#### 2.12.4.4 Các vấn đề chú ý và mở rộng khi dùng vòng lặp for

- Các biến trong biểu thức `statement` của `for` sẽ được sử dụng trong phạm vi của `for`, chúng sẽ được giải phóng và không tồn tại khi kết thúc (ra khỏi) vòng lặp `for`.
- Các biến được khai báo trong khôi lệnh của `for` sẽ được sử dụng trong phạm vi của 1 vòng lặp `for`, chúng sẽ được giải phóng và khai báo biến mới ở mỗi vòng lặp và sẽ không tồn tại khi kết thúc (ra khỏi) vòng lặp `for`.
- Trong Khôi lệnh vòng lặp for và while chúng ta nên hết sức tránh khai báo biến hoặc cấp phát bộ nhớ cho biến vì sẽ rất nhanh tiêu hao bộ nhớ qua mỗi vòng lặp.
- Chúng ta có thể sử dụng vòng for theo các cách khác thường nhưng vẫn hợp lệ như sau:

- o Kiểu 1 : Thiếu vắng `statement`

```

14 |   int i = 0;
15 |   for(; i< 5; i++){
16 |     System.out.println("Round and round we go " + i);
17 |

```

- o Kiểu 2 : `statement` là một phép cộng

```

14 |   int i = 0;
15 |   for( i=i+1; i< 5; i++){
16 |     System.out.println("Round and round we go " + i);
17 |

```

- o Kiểu 3 : `statement` bao gồm 2 biểu thức cộng.

```

14 |   int i = 0, j = 0;
15 |   for( i++, j++; i+j < 5; i++){
16 |     System.out.println("Round and round we go " + i);
17 |

```

- Tuy nhiên chúng ta lại không thể khai báo như sau:

```
for ( int i = 0, int j = 0; i+j < 5; i++ )
```

```
for (int i = 0; i = i + 1; i + j < 5; i++ )
```

- Chúng ta có thể khai báo 2 kiểu nguy hiểm sau:

```
47 |     for (;;) {
48 |         System.out.println("Round and round we go for ever.");
49 |     }
```

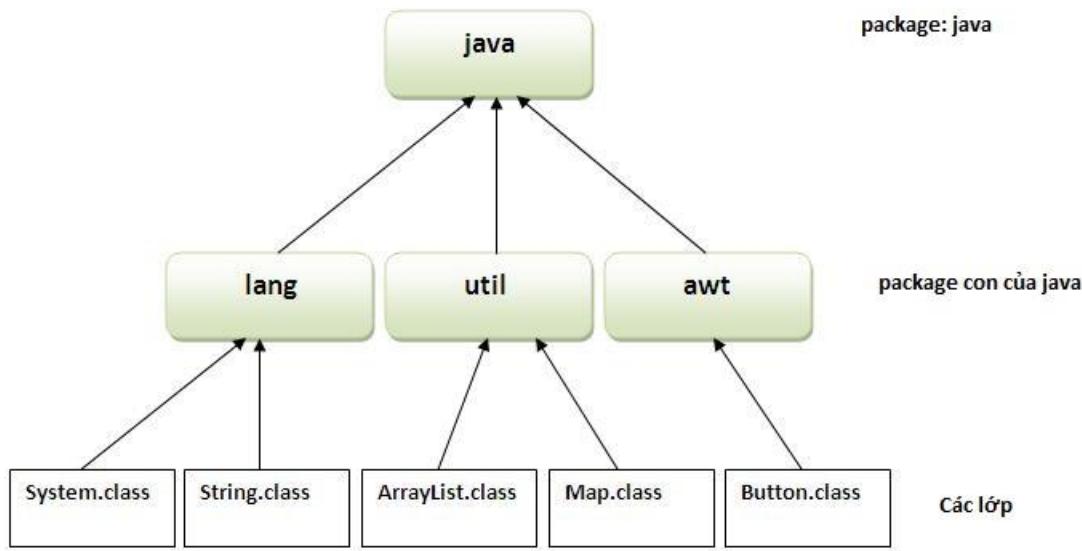
Nhưng chúng ta lại không thể khai báo như sau:

```
47 |     for () {
48 |         System.out.println("Round and round we go for ever.");
49 |     }
50 | }
```

- ✓ Vậy là chúng ta đã nghiên cứu xong về cấu trúc điều khiển lặp For.

## 2.13 Nghiên cứu sâu về Classes trong Java

### 2.13.1 Package và cách tổ chức thư mục Trong Java



Để khai báo một lớp thuộc một package thì ta dùng từ khóa “**package**” theo cú pháp sau:

```
package <PackageName>;
```

Ví dụ như hình bên dưới khai báo Package có tên **jb12** và file Welcome.java phải nằm trong thư mục **jb12** tương ứng trong thư mục chứa source code của dự án.

```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package jb12;
7
8  import java.util.Scanner;
9
10 /**
11 *
12 * @author Tanth
13 */
14 public class Welcome {
15
16     public static void main(String[] args) {

```

- Ví dụ về việc các package được tổ chức theo như phân cấp cây thư mục. Hình bên dưới là tổ chức các package lồng nhau như sau:

```

1 com.activestudy.utility
2   com.activestudy.utility.Define
3   com.activestudy.utility.file
4   com.activestudy.utility.lic
5   com.activestudy.utility.parse
6   com.activestudy.utility.system
7   com.activestudy.utility.db
8     com.activestudy.utility.db.oracle
  
```

```

4  /*
5  *  dinh open the template in the editor
6  */
7
8 package com.activestudy.utility;
9
10 import java.io.FileInputStream;
11 import java.io.FileOutputStream;
12 import java.util.Date;
13 import java.util.Properties;
  
```

- o Package cha: `com`
- o Package con: `activestudy`
- o Package cháu: `utility`

Hình bên dưới là một ví dụ về khai báo lớp `RuntimeThread` thuộc Package `com.activestudy.utility`

```

1 /**
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6 package com.activestudy.utility;
7
8 import org.apache.commons.logging.Log;
9
10 /**
11  *
12  * @author tanhai
13  */
14 public class RuntimeThread extends ActionThread{
15     Log logger = null;
16     long sleepTime = 60000;
  
```

- **Package** Là một thành phần để đóng gói các file code Java và đóng gói luôn cả các Package con bên trong nó.
- Các lớp và các Interface có cùng đặc điểm hoặc chức năng chung thì được chứa đựng vào cùng một Package tùy theo tổ chức của lập trình viên.
- Package tạo ra một phạm vi về truy cập, sự truy cập đến các lớp, thuộc tính và hành vi của đối tượng chịu sự chi phối của các Access Modifier và Package. Chúng ta sẽ tìm hiểu điều này ở phần sau khi nghiên cứu Access Modifier.
- Package khắc phục được việc đặt trùng tên vì các clas được chia ra ở các package khác nhau.

- Chú ý khi dùng tới các class thuộc package nào thì ta mới import class đó vào package như sau, tránh import quá nhiều class không sử dụng đến sẽ làm tăng kích thước chương trình java của chúng ta. Vị trí khai báo sẽ nằm sau khai báo package và trên khai báo class.

```

7  package jb12;
8
9  import com.activestudy.utility.RuntimeThread;
10
11 /**
12  * @author Tanh
13  */
14 public class JB12 {

```

- Chú ý nếu class ta dùng cùng thuộc một package với class hiện tại thì không cần thiết phải import.
- Ta cũng có thể không cần import nếu class ta dùng chỉ một hoặc 2 lần và ta có thể sử dụng trực tiếp class nhưng thay vì viết mỗi tên trong quá trình khai báo biến thì ta cần chỉ ra cả package và tên class. Ví dụ:

```

6  package jb12;
7  //import com.activestudy.utility.RuntimeThread;
8
9  /**
10  *
11  * @author Tanh
12  */
13  public class JB12 {
14  /**
15  * @param args the command line arguments
16  */
17  public static void main(String[] args) {
18      com.activestudy.utility.RuntimeThread runtime;           Variable runtime is not used
19  }
20 }

```

- Ta cũng có thể import nhiều class của một package bằng một import dạng như ví dụ sau:

```

? import com.activestudy.utility.*;
8 /**
9 *
10 * @author Tanh
11 */
12 public class JB12 {
13 /**
14 * @param args the command line arguments
15 */
16 public static void main(String[] args) {
17     RuntimeThread runtime;
18 }
19 }

```

- Trong thực tế tên của package được tổ chức như sau:

- o Phần đầu tiên thể hiện phạm vi của Package ví dụ như sau:
    - `com` thường là các package có mục đích thương mại và được các công ty, tổ chức phát triển phần mềm thương mại sử dụng, trước khi sử dụng cần có sự cho phép của nhà phát triển.
    - `org` thường là các package chứa mã nguồn mở của cộng đồng.
- `com.activestudy.utility;`
- `org.activestudy.utility;`
- o Phần thứ 2 thể hiện tên của công ty, tổ chức. ví dụ: `com.activestudy.utility`; Trong đó `activestudy` là tên tổ chức.
  - o Phần thứ 3 thường thể hiện tên của dự án
  - o Các phần sau sẽ đặt theo tên chức năng của package.

✓ Vậy là ta đã hoàn thành nghiên cứu về Package và cách tổ chức thư mục Trong Java

### 2.13.2 Khai báo Class (Declaring Classes) trong Java

- Kể từ khi bắt đầu viết chương trình “Hello world” đầu tay chúng ta đã bắt đầu biết đến class. Chúng ta đã viết code trong class như hình bên dưới, chắc các bạn còn nhớ. Bây giờ chúng ta đi tìm hiểu kỹ hơn về class, các kỹ thuật lập trình với class.

```
public class Welcome {
    public static void main(String args[]) {
        System.out.println("Welcome to Java program");
    }
}
```

- **Class (Lớp):** Là một kiểu dữ liệu có sẵn trong bộ JDK, đó là những lớp mô tả kiểu dữ liệu mà chúng ta hay dùng như `String`, `Integer`, `Scanner`... hoặc do lập trình viên tự thiết kế và class cũng chính là bản thiết kế của các đối tượng (Object), trong mỗi class có các thuộc tính(variable) và phương thức (Method) thể hiện chức năng của đối tượng.
- **Cú pháp khai báo một class như sau:**

`<Access modifier> class <ClassName>`

```
{
    //class body
}
```

- Trong đó
  - o **Access modifier** chỉ chấp nhận 2 loại đó là **public** hoặc **default**(default là kiểu để trống ko điền acces moidifier, nó khác với public, protected và private).
  - o **class** là từ khóa khai báo class.
  - o **ClassName** là tên của class được đặt trong lúc lập trình thiết kế class. Chú ý Nếu class có access modifier là public thì tên của class này phải trùng với tên file chứa nó. Vì vì vậy mà là trong một file chỉ chứa 1 class có thuộc tính public duy nhất mà thôi.
  - o **Class body:** là nội dung chính của class, nằm trong dấu {/body here} của class, Trong đó để khai báo các biến (thuộc tính), các phương thức (method), phương thức khởi dụng.
- Sau đây là ví dụ về class:

```

11  public class Person {
12      private String name = "sophia";
13      private int high;
14      private int weight;
15      private int yearOfBirth;
16      private boolean isMarried;
17
18      public Person(String name){
19          this.name = name;
20          this.isMarried= false;
21      }
22      public void speakName(String name){
23          this.name = name;
24          String say_ = "Hello, I am robot, my name is " + name;
25          System.out.println(say_);
26      }
27 }
```

- Xét ví dụ trên ta có tên class là **Person**, các thuộc tính **name**, **high**, **weight**, **yearOfBirth**, **isMarried**. Class cũng có phương thức **speakName**, và đặc biệt là phương thức khởi dụng **Person** (Cứ phương thức nào có cùng tên với class thì được gọi là phương thức khởi dụng và phương thức khởi dụng sẽ được chúng ta tìm hiểu ở phần tiếp theo).
- ✓ Vậy là chúng ta đã tìm hiểu xong về khai báo một class trong java.

### 2.13.3 Khai báo biến thành viên (Declaring Member Variables) trong Java

- Việc khai báo các biến thành viên hay còn gọi là thuộc tính hoặc trường dữ liệu trong class cũng chính là công việc khai báo biến bình thường ta hay dùng. Ta đi vào chi tiết việc khai báo như sau:

<Access modifier> <tiền tố> <Type> <varName>;

Trong đó

- **Access modifier** : Quy định phạm vi truy cập của biến, ta sẽ tìm hiểu kỹ về điều này ở phần sau về Access modifier. Các access modifier được phép dùng như sau: **public, protected, default, private**.
- **Tiền tố**: có các tiền tố quy định thêm về tính chất của thuộc tính. Ví dụ như sau:

- o **static** :

- Ta xét ví dụ sau

```

11 public class Person {
12     private String name = "sophia";
13     private int height;
14     private int weight;
15     private int yearOfBirth;
16     private boolean isMarried;
17     private static int nCounter; //Total of person.
18     public Person(String name) {
19         this.name = name;
20         this.isMarried= false;
21     }
22     public int increatCouter(){
23         return ++nCounter;
24     }
25     public void speakName(String name) {
26         this.name = name;
27         String say_ = "Hello, I am robot, my name is " + name;
28         System.out.println(say_);
29     }
30 }
```

- Từ khóa **static** chỉ ra đây là một biến tĩnh trong class, trong hình dưới đây là khai báo **nCounter** trong class **Person**.

```
17     public static int nCounter; //Total of person.
```

Và **nCounter** được sử dụng trực tiếp bằng cách gọi biến từ tên của class mà không cần phải khởi tạo đối tượng thuộc lớp **Person**.

```

20     public static void main(String[] args) {
21         Person.nCounter += 1;
22         System.out.println("Total of Persons: " + Person.nCounter);
23 }
```

Sở dĩ có được điều này là vì tất cả các biến static đều đã được khai báo và cấp phát bộ nhớ trên static memory ngay khi JVM khởi chạy chương trình, vì vậy các biến tĩnh tồn tại độc lập với các đối tượng và có thể được truy cập ngay cả khi các đối tượng chưa hề được khai báo.

- Ngoài ra (Trong cùng một thread) khi chỉ thị là `static` cho một biến thì tất cả các đối tượng khai báo của class đó đều chỉ có duy nhất 1 biến thành viên cho tất cả các đối tượng này khi đối tượng được khai báo và khởi tạo. Nó tương đương khái niệm là chỉ có 1 ô nhớ duy nhất trên bộ nhớ static được cấp phát làm nơi lưu trữ giá trị của biến static đó.

```

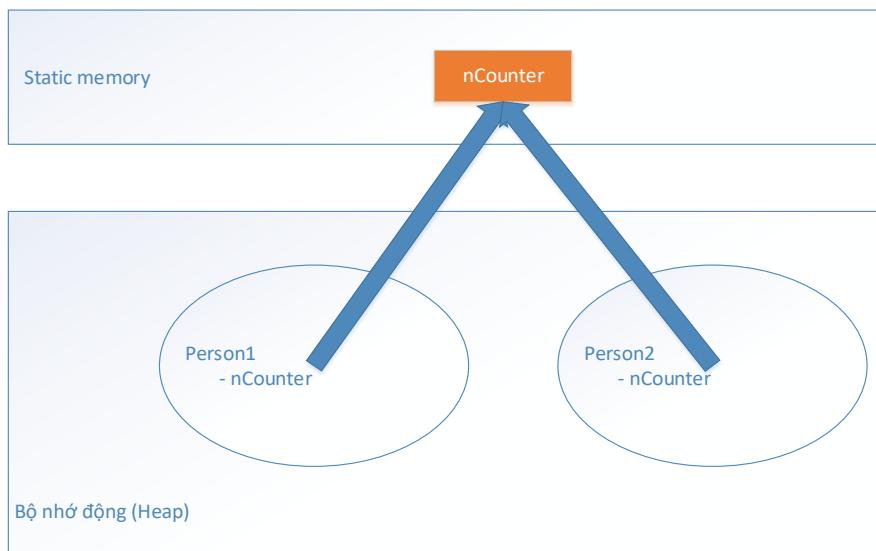
24     Person person1 = new Person();
25     Person person2 = new Person();
26
27     System.out.println("Total of Persons: " + Person.nCounter);
28     System.out.println("Total of Persons: " + person1.nCounter);
29     System.out.println("Total of Persons: " + person2.nCounter);
30     person1.nCounter +=1;
31     System.out.println("Total of Persons: " + Person.nCounter);
32     System.out.println("Total of Persons: " + person1.nCounter);
33     System.out.println("Total of Persons: " + person2.nCounter);
  
```

Kết quả sau khi chạy:

```

Output - JavaCoreBasic (run) ×
run:
Total of Persons: 1
Total of Persons: 2
Total of Persons: 2
Total of Persons: 2
  
```

Chúng ta xem hình sau đây sẽ minh họa cho cơ chế của biến static:



- o **final:** Tiền tố chỉ ra rằng một biến có giá trị không thể thay đổi sau khi được khai báo và khởi tạo giá trị trong lần đầu tiên. Biến kiểu này có vùng nhớ thuộc vùng nhớ constant trong bộ nhớ máy tính, ta còn gọi là biến hằng hay hằng số.

```

14  public class Person {
15      private String name = "sophia";
16      private int high;
17      private int weight;
18      private int yearOfBirth;
19      private boolean isMarried;
20      private static int nCounter; //Total of person.
21      private final int MAX_PERSON = 10000;
22
23      public Person(String name) {
24          this.name = name;
25          this.isMarried= false;
26      }
27      public int increatCouter() {
28          if(nCounter < MAX_PERSON) {
29              return ++nCounter;
30          }
31          return MAX_PERSON;
32      }

```

Biến hàng được sử dụng khi ta muốn sử dụng biến số với một giá trị cố định ví dụ như một tiêu chuẩn hoặc một giới hạn không thể thay đổi như ví dụ trên.

- ✓ Vậy là chúng ta đã nghiên cứu xong về kỹ thuật khai báo biến thành viên trong class java.

#### 2.13.4 Khai báo phương thức (Defining Methods) trong Java

- Chúng ta đã sử dụng phương thức và đã làm quen với nó, bây giờ chúng ta đi tìm hiểu kỹ hơn về phương thức như cách khai báo và các kỹ thuật mở rộng khác. Sau đây là phương thức increatCouter() trong class Person.

```

55      public int increatCouter() {
56
57          if (nCounter < MAX_PERSON) {
58              return ++nCounter;
59          }
60          return MAX_PERSON;
61      }

```

- Phương thức (Method) là một thành phần chứa đựng các đoạn mã code để thực hiện chức năng logic nào đó. Phương thức có tên gọi, kiểu trả về và các tham số đầu vào. Phương thức là một

trong những phương tiện để quy hoạch các chức năng thể hiện hành vi của đối tượng của class.  
Sau đây xem xét tới cú pháp khai báo một phương thức cụ thể:

```
<Access modifier>_<Tiền tố>_<kiểu trả về>_<Tên phương thức> (<danh sách tham số>) {  
  
<khoi lệnh>;  
  
}
```

Trong đó:

- **Access modifier** : quy định khả năng truy cập tới method của đối tượng.
- **Tiền tố**: có các tiền tố quy định thêm về tính chất của thuộc tính. Ví dụ như sau:
  - o **Static** : Cũng giống như đối với thuộc tính, Phương thức tĩnh là phương thức có thể được sử dụng ngay khi chưa cần khai báo và khởi tạo một đối tượng thuộc lớp.

```
14  public class Person {  
15      private String name = "sophia";  
16      private int high;  
17      private int weight;  
18      private int yearOfBirth;  
19      private boolean isMarried;  
20      private static int nCounter; //Total of person.  
21      private final int MAX_PERSON = 10000;  
22  
23      public Person(String name){  
24          this.name = name;  
25          this.isMarried= false;  
26      }  
27      public int increatCouter(){  
28          if(nCounter < MAX_PERSON){  
29              return ++nCounter;  
30          }  
31          return MAX_PERSON;  
32      }  
33      public static int getCounter(){  
34          return nCounter;  
35      }  
Ví
```

Ví dụ về truy cập tới phương thức tĩnh.

```
16  public static void main(String[] args) {  
17      System.out.println("Total of Person: " + Person.getCounter());  
18  }
```

**Chú ý.** Trong một phương thức tĩnh khi viết code ta chỉ có thể khai báo và sử dụng những biến local của phương thức hoặc truy cập tới những biến và phương thức tĩnh khác chứ không thể sử dụng các biến và phương thức bình thường được.

- **abstract:**

- là tiền tố chỉ ra phương thức trừu tượng, phương thức này không cần phải có body (phần mã code) và phần body sẽ được viết lại trong các lớp kế thừa nó (lớp dẫn suất của nó), như vậy là các lớp nào nàò khi đã kế thừa thì cần phải khai báo và viết lại (override) phương thức abstract nếu không thì nó cũng phải là lớp abstract.
- Lớp chứa đựng các phương thức abstract thì cũng là lớp abstract. Chúng ta sẽ đi tìm hiểu về lớp abstract ở phần sau.
- Ta xét ví dụ sau về phương thức abstract speakName, bởi vì lớp Person mô tả lớp con người chung chung nên sẽ không biết con người đó nói ngôn ngữ gì, vì ta biết rằng mỗi một quốc gia có ngôn ngữ riêng, do vậy mỗi một lớp mô tả con người của quốc gia khi kế thừa từ lớp Person sẽ phải override lại phương thức speakName theo đúng như ngôn ngữ ở quốc gia của mình.

```

14  public abstract class Person {
15      private String name = "sophia";
16      private int height;
17      private int weight;
18      private int yearOfBirth;
19      private boolean isMarried;
20      private static int nCounter; //Total of person.
21      private final int MAX_PERSON = 10000;
22
23      public Person(String name){
24          this.name = name;
25          this.isMarried= false;
26      }
27      public abstract void speakName(String name);
28

```

- **synchronized:** là từ khóa được sử dụng trong multithread để nhằm đồng bộ việc truy cập tới các method của cùng một đối tượng, tránh các truy cập đồng thời dẫn tới xung đột giữa các thread. Phần này ta sẽ có nghiên cứu sâu về quá trình đồng bộ này trong phần Multithread (đa luồng) của Java core advance.
- ✓ Vậy là chúng ta đã hoàn thành tìm hiểu về cách xây dựng một phương thức trong class và một số đặc điểm của nó.

### 2.13.5 Phương thức khởi dựng (constructor method)

- Phương thức khởi dựng là phương thức có cùng tên với class và không có kiểu trả về vì kiểu trả về chính là kiểu của class đó.

Ta lại xét ví dụ về class Person có 2 phương thức khởi dựng có cùng tên, một phương thức có và một phương thức không có tham số.

```

①  public abstract class Person {
15    protected String name = "sophia";
16    protected int high;
17    protected int weight;
18    protected int yearOfBirth;
19    protected boolean isMarried;
20    private static int nCounter; //Total of person.
21    private final int MAX_PERSON = 10000;
22    Children children;
23    public Person(){
24        children = new Children(this);
25    }
26    public Person(String name){
27        this.name = name;
28        this.isMarried= false;
29    }

```

- Phương thức khởi dựng không tham số là phương thức khởi dựng mặc định, không cần khai báo cũng có.
- **Chú ý:** Khi đã khai báo phương thức khởi dựng có tham số thì phương thức khởi dựng mặc định sẽ không còn tồn tại nữa, lúc này nếu muốn sử dụng phương thức khởi dựng không tham số thì ta cần phải khai báo thêm.  
Ở ví dụ trên. Phương thức khởi dựng `public Person(String name)` là phương thức khởi dựng có tham số, và khi khai báo phương thức này thì phương thức khởi dựng mặc định không tham số biến mất, vì vậy ta khai báo thêm một phương thức khởi dựng không tham số để sử dụng khi cần. `public Person()`.
- Phương thức khởi dựng được dùng và chạy đầu tiên khi khởi tạo đối tượng bằng từ khóa `new`.

```

17     Person person1 = new Person("Nam") ;
18     Person person2 = new Person() ;
19

```

Do được chạy đầu tiên nên phương thức có tên là khởi dựng và thường được sử dụng vào mục đích khởi tạo các giá trị ban đầu cho đối tượng.

- Phương thức khởi dựng cũng có thể được gọi ra ở lớp con kế thừa (lớp dẫn suất) của nó bằng từ khóa **super()** ở dòng đầu tiên của phương thức khởi dựng của lớp con kế thừa của nó.
- Phương thức khởi dựng cũng có thể được gọi ra ở đầu của một phương thức khởi dựng khác thông qua từ khóa **this**.

```

23  public Person() {
24      this("sophia");
25      children = new Children(this);
26  }
27  public Person(String name) {
28      this.name = name;
29      this.isMarried= false;
30  }

```

- ✓ Vậy là chúng ta đã đi tìm hiểu xong về phương thức khởi dựng.

#### 2.13.6 Đối tượng(Object) là gì, phân biệt giữa đối tượng (Object) và Class trong Java.

- Xét ví dụ sau:

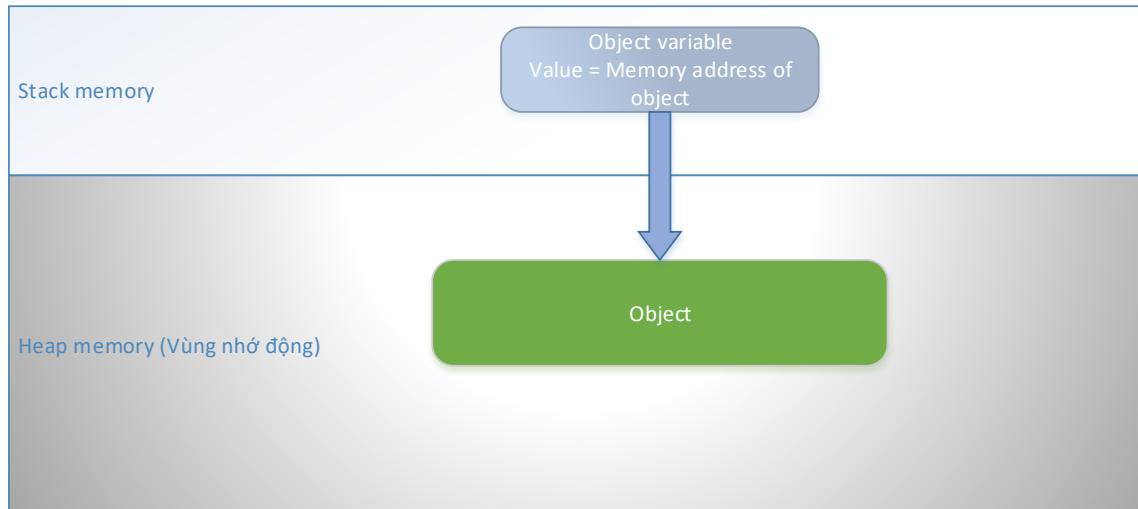
```
24 | Person person1 = new Person();
```

Thì

- o **Person** là class
- o **person1** là biến đối tượng
- o Tại dòng 24 một đối tượng được khởi tạo bằng từ khóa **new** và được biến đối tượng **person1** trả tới. Hay nói cách khác biến đối tượng **person1** nắm giữ giá trị địa chỉ ô nhớ của đối tượng khởi tạo tại dòng 24.

- Đối tượng là các thực thể được khai báo từ các class và được đại diện bởi các biến đối tượng (còn gọi là biến Reference).
- Trong khi class được ví như bản thiết kế của một thực thể ví dụ như bản thiết kế của ngôi nhà thì đối tượng là một ngôi nhà được xây dựng dựa trên bản thiết kế là class.
- Class bản thân nó không có không gian nhớ và không thể thực hiện được các phương thức (trừ phương thức tĩnh static) thì đối tượng là các thực thể được cấp phát không gian nhớ (trên heap memory) và có thể thực thi được các hành vi của nó đã được thiết kế bởi class.

- Đối tượng khi được khai báo thì chương trình sẽ cấp phát một không gian nhớ trên bộ nhớ (Heap memory) và nó được đại diện bởi một biến object (Biến reference) nằm trên vùng nhớ stack. Muốn truy cập tới các đối tượng thì ta sử dụng các biến đối tượng đã nắm giữ (trỏ tới) địa chỉ ô nhớ của đối tượng đó.
- Ta xét hình minh họa sau đây để thấy được điều này.



✓ Vậy là ta đã phân biệt được các khái niệm Class, Biến đối tượng và đối tượng là khác nhau.

### 2.13.7 Từ khóa this là gì và sử dụng thế nào

- Từ khóa this là một biến mặc định của một đối tượng khi được khai báo, nó được trả tới chính đối tượng đó và chỉ được đối tượng đó sử dụng ở trong nội tại của nó. Sau đây ta xét ví dụ cụ thể để thấy được điều này:

```

14  public class Person {
15      private String name = "sophia";
16      private int height;
17      private int weight;
18      private int yearOfBirth;
19      private boolean isMarried;
20      private static int nCounter; //Total of person.
21      private final int MAX_PERSON = 10000;
22
23      public Person(String name){
24          this.name = name;
25          this.isMarried= false;
26      }
    ...
  
```

- Dòng số 24 và 25 sử dụng Từ khóa this để trả lời thuộc tính được khai báo trong class (biến instance). Trong trường hợp này ta sử dụng this để nhằm tưởng minh cho việc chỉ ra thuộc tính name là của class chứ không phải là tham số name truyền vào từ hàm. Từ khóa this không thể dùng để truy cập tới các biến cục bộ (biến local của method).
- Tương tự để truy cập tưởng minh tới các phương thức của lớp thì ta có thể thêm Từ khóa this ở đầu.
- Từ khóa this có thể được truyền như là một tham số trong lời gọi phương thức, phương thức khởi dụng.

```

⑧ public abstract class Person {
15     protected String name = "sophia";
16     protected int height;
17     protected int weight;
18     protected int yearOfBirth;
19     protected boolean isMarried;
20     private static int nCounter; //Total of person.
21     private final int MAX_PERSON = 10000;
22     Children children;
23     public Person() {
24         children = new Children(this);
25     }

```

- Từ khóa this còn để gọi một phương thức khởi dụng khác ở trong một phương thức khởi dụng.

```

23     public Person(){
24         this("sophia");
25         children = new Children(this);
26     }
27     public Person(String name){
28         this.name = name;
29         this.isMarried= false;
30     }

```

## 2.14 Access Modifiers Trong Java

- Xem xét bảng mô tả về khả năng truy cập đến các thuộc tính và phương thức của một class và đối tượng như sau:

Access Modifier	Nội bộ Class	Nội bộ Package	Nội bộ Subclass	Other
<b>private</b>	Yes	No	No	No
<b>Default</b>	Yes	Yes	No	No
<b>protected</b>	Yes	Yes	Yes	No
<b>public</b>	Yes	Yes	Yes	Yes

### 2.14.1.1 Public

- Một thuộc tính hoặc phương thức của một class được khai báo Access modifier là public thì nó có thể được truy cập từ mọi nơi trong chương trình java, ví dụ như từ các lớp kế thừa cùng package, từ các đối tượng cùng package...
- Ta nên hạn chế khai báo một biến có access modifier là public để ngăn chặn sức truy cập trực tiếp tới chúng. Dữ liệu nên được truy cập thông qua các phương thức getter và setter (get/set).

```

45  ┌─────────┐
46  │         │
47  │   public String getName() {
48  │   │         return name;
49  │   └─────────┘
50  ┌─────────┐
51  │         │
52  │   public void setName(String name) {
53  │   │         this.name = name;
54  │   └─────────┘

```

### 2.14.1.2 Protected

- Phương thức hoặc biến có access modifier là protected có thể được truy cập từ các lớp con kế thừa hoặc trong cùng package.

```

6   package jb12;
7   ┌─────────┐
8   │         │
9   │   /**
10  │   *      *
11  │   * @author Tanht
12  │   */
13  ┌─────────┐
14  │         │
15  │   public class Person {
16  │       protected String name = "sophia";

```

Lớp Person thuộc package **jb12** và lớp Children kế thừa từ lớp **Person** nhưng class **Person** lại thuộc package **other**, vì vậy trong nội bộ lớp **Children** cũng không thể truy cập tới các thuộc tính **Protected** được.

```

6   package other;
7
8   import jbl2.Person;
9
10  /**
11   * @author tanth
12  */
13  public class Children extends Person{
14      public Children(Person person) {
15          this.name = person.name;
16      }

```

#### 2.14.1.3 Default

- Khi thuộc tính hoặc phương thức trong class không được chỉ ra một access modifier thì có nghĩa là nó có access modifier mặc định (default).
- Default quy định thuộc tính và phương thức chỉ được truy cập trong phạm vi cùng class hoặc các lớp kế thừa nhưng cùng phạm vi một package.
- Nếu truy cập các thuộc tính hoặc phương thức thông qua đối tượng thì đối tượng phải cùng thuộc một package với class đó.

```

6   package jbl2;
7   /**
8   *
9   * @author Tanth
10  */
11  @
12  public class Person {
13      String name = "sophia";

```

Do cùng thuộc package jbl2 nên trong lớp Children và thông qua biến đối tượng person thì ta đều có thể truy cập được tới thuộc tính name.

```

6   package jbl2;
7
8   /**
9   *
10  * @author tanth
11  */
12  public class Children extends Person{
13
14      public Children(Person person) {
15          this.name = person.name;
16          this.high = person.high;

```

Do lớp Children thứ 2 được khai báo trong package other nên không cùng package với lớp Person vì vậy cho dù là lớp kế thừa và đối tượng thì cũng không thể truy cập được đến thuộc tính name của lớp cha.

```

6   package other;
7
8   import jbl2.Person;
9
10  /**
11   *
12   * @author tanth
13   */
14  public class Children extends Person {
15
16      public Children(Person person) {
17          this.name = "";
18          person.name = "";
19          this.height = person.height;
20          this.weight = person.weight;
21          this.yearOfBirth = person.yearOfBirth;
22      }

```

#### 2.14.1.4 Private

- Ngăn cản mọi truy cập trực tiếp đến các thuộc tính và phương thức từ bên ngoài class khai báo nó, thậm chí là từ các class kế thừa từ nó cho dù là cùng một package.
- Private nên được sử dụng cho các thuộc tính để đảm bảo tính an toàn dữ liệu cho các đối tượng.

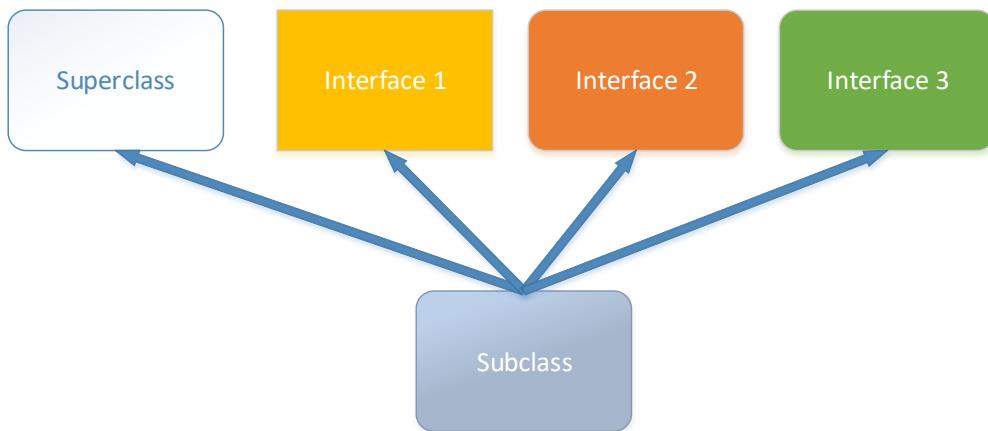
### 2.15 Lớp kế thừa trong Java

- Để tiết kiệm thời gian và chi phí viết mã code thì việc kế thừa là một khả năng rất quan trọng. Nó cho phép lập trình viên sử dụng tiếp class đã viết để phát triển tiếp class sau. nhưng điều này là dựa trên những sự phù hợp logic về dữ liệu và phương thức thì mới nên làm điều này, ta không thể thiết kế một class kế thừa từ một class không liên quan, ví dụ như class mô tả về một ngôi nhà House không thể kế thừa từ class Person mô tả về con người.
- Để thiết kế một class kế thừa từ lớp khác ta dùng từ khóa extends

```
12 |     public class Children extends Person{
```

- Class kế thừa gọi là subclass, derived class, child class, extended class hay còn gọi là lớp dẫn suất, lớp cho kế thừa hay còn gọi là lớp cha, base class, Parent class, superclass.

- Khi thiết kế các đối tượng chúng ta cần đặt ra 2 câu hỏi, thứ nhất liệu class này có thể kế thừa từ lớp nào không, Câu hỏi thứ 2 lớp này liệu sẽ có khả năng cho lớp khác kế thừa không. Khi làm rõ được 2 vấn đề này thì ta đi thiết kế class sẽ theo đúng hướng của lập trình hướng đối tượng hơn và tiết kiệm được thời gian phát triển hơn, chương trình sẽ tinh gọn hơn.
- Một class chỉ có thể được kế thừa từ 1 class cha, nhưng lại được kế thừa từ nhiều Interface khác nhau, người ta gọi là kế thừa lai (*hybrid*). Đối với interface ta nên gọi là implement thay vì gọi là kế thừa.



- Vậy tại sao một class chỉ có thể kế thừa từ 1 class cha. Ta hãy hình dung nếu class con kế thừa từ 2 class cha, hai class cha này có một method cùng tên và cùng tham số đầu vào thì trình thông dịch biết gọi method của class cha nào để dùng đây. Điều này gây lên mập mờ trong việc gọi method và gây lên sai logic. Vì vậy Java ngăn chặn việc này bằng cách chỉ kế thừa từ 1 class.
- Class con chỉ được kế thừa những gì lớp cha cho phép thông qua các Access modifier, kế thừa ở đây được hiểu là được sử dụng và truy cập. Những thứ mà lớp cha có thể cho phép kế thừa đó là các thuộc tính, các phương thức của lớp cha. Cụ thể là những thuộc tính và phương thức có access Modifier Public, protected, default thì các lớp con cùng thuộc một package đều có thể kế thừa lại. Nếu khác package thì chỉ có public và protected mới được kế thừa
- Class con cũng giống như các class khác và nó có thể được thiết kế với các thuộc tính và phương thức của riêng nó, và tất nhiên lớp cha không thể sử dụng được những thuộc tính và phương thức của lớp con vì lớp cha không kế thừa từ lớp con, Chúng ta không thể thiết kế các class kế thừa vòng tròn được vì điều này là nghịch lý, không thuận với logic tự nhiên trong cuộc sống và xã hội. như ví dụ dưới là không thể do class Children đã kế thừa từ Person và Person không thể kế thừa lại của Children được nữa.



```
public class Person extends Children{
```

Xét ví dụ sau về class Children kế thừa từ lớp Person

- Nếu muốn không cho class nào có thể kế thừa lại được nữa thì khi khai báo class thêm từ khóa **final** vào đầu tiên.

```
① public final class Person {
12   public class Children extends Person{
13
14     public Children(Person person) {
15       this.name = person.name;
16       this.high = person.high;
17       this.weight = person.weight;
18       this.yearOfBirth = person.yearOfBirth;
19     }
}
```

Class **Children** sử dụng các thuộc tính **name**, **high**.. như của nó.

## 2.16 Overloading trong Java

- Trong khi phát triển các phương thức của class, ta nhận thấy có nhiều phương thức làm cùng một nhiệm vụ gần giống nhau và trả ra cùng thậm chí khác nhau về kiểu dữ liệu. Chúng chỉ khác nhau ở kiểu hoặc số lượng tham số đầu vào mà thôi. Lúc này ta có thể viết thành các phương thức khác nhau nhưng lại có cùng tên, điều này cho phép chúng ta dễ dàng trong việc đặt tên khi thiết kế và triệu hồi các phương thức hơn khi sử dụng. Kỹ thuật này được gọi là overloading trong lập trình.
- Tất cả các phương thức đều có thể được overload kể cả phương thức khởi và phương thức hủy.

```
20   public Person() {
21     this("sophia");
22     children = new Children(this);
23   }
24   public Person(String name) {
25     this.name = name;
26     this.isMarried= false;
27 }
```

- Các phương thức overload thì phải cùng tên và khác nhau về kiểu dữ liệu của tham số đầu vào hoặc số lượng của tham số đầu vào.
- Ta đi xét ví dụ về một method tính max được overload theo 3 phương thức. Và với 3 phương thức này thì method max() có khả năng xử lý tìm max() được cả 2 số nguyên int, 2 số thực double và 3 số thực double.

```

24 	/** Return the max of two int values */
25 	public int max(int num1, int num2) {
26 		if (num1 > num2) {
27 			return num1;
28 		} else {
29 			return num2;
30 		}
31 	}
32 	/**
33 	* Find the max of two double values
34 	*/
35 	public double max(double num1, double num2) {
36 		if (num1 > num2) {
37 			return num1;
38 		} else {
39 			return num2;
40 		}
41 	}
42 	/**
43 	* Return the max of three double values
44 	*/
45 	public double max(double num1, double num2, double num3) {
46 		return max(max(num1, num2), num3);
47 	}

```

- Khi triệu gọi một phương thức đã được overload thì trình biên dịch sẽ căn cứ vào các tham số đầu vào để tự đi tìm xem phương thức nào là phù hợp. nếu không tìm thấy phương thức nào là phù hợp thì trình biên dịch sẽ báo lỗi không tìm thấy phương thức như khi chúng ta gọi sai tên method.
- Trường hợp máy tính không thể phân biệt được phương thức nào khi triệu gọi khi có sự mập mờ về dữ liệu đầu vào như sau:

```

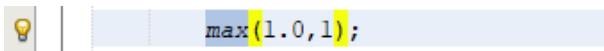
42 	public static double max(int num1, double num2) {
43 		if (num1 > num2) {
44 			return num1;
45 		} else {
46 			return num2;
47 		}
48 	}
49 	public static double max(double num1, int num2) {
50 		if (num1 > num2) {
51 			return num1;
52 		} else {
53 			return num2;
54 		}
55 	}

```

Khi triệu gọi sẽ bị báo lỗi Ambiguous bởi vì 2 số 1 được hiểu là int và nó không biết nên cast tham số 1 hay tham số 2 sang kiểu double để gọi bởi vì cả 2 đều có thể tìm được một method tương ứng, vì vậy để tránh sai sót về mặt logic thì chúng ta nên tránh trường hợp mập mờ về kiểu dữ liệu này.



Nên chỉ rõ kiểu dữ liệu đầu vào như sau để xác định chính xác phương thức cần triệu gọi:



## 2.17 Overriding trong Java

- Nói đến overriding là nói đến việc lớp con muốn viết lại hành vi trong method của lớp cha hoặc là bổ sung hành vi vào method abstract của lớp cha hoặc là thực thi các method của interface mà nó kế thừa.
- Khi class con kế thừa method của lớp cha mà lớp con muốn xử lý logic method này theo cách riêng của mình thì lớp con có thể viết lại method này và khai báo thêm annotation @override ở phía trên của method.

```

9  * @author Tanth
10 */
11 public class Person {
12     String name = "sophia";
13     protected int high;
14     protected int weight;
15     protected int yearOfBirth;
16     protected boolean isMarried;
17     private static int nCounter; //Total of person.
18     private final int MAX_PERSON = 10000;
19
20     public String say() {
21         String text = "Hello";
22         System.out.println(text);
23         return text;
24     }

```

Lớp con là VietNamPerson kế thừa từ lớp cha Person muốn override lại phương thức say của lớp cha Person để nói theo kiểu người Việt Nam thì lớp con sẽ cài đặt như sau:

```

10  * @author tanth
11 */
12 public class VietNamPerson extends Person{
13     @Override
14     public String say() {
15         String text = "Xin Chao";
16         System.out.println(text);
17         return text;
18     }
19 }

```

- Một đối tượng thuộc kiểu VietNamPerson khi triệu gọi phương thức say thì kết quả sẽ hiển thị ra chữ “Xin chao” thay vì hiển thị “Hello” như lớp cha Person của nó.

```

14 public class Welcome {
15
16     public static void main(String[] args) {
17         Person personl = new Person("Nam");
18         personl.say();
19         VietNamPerson vietNam = new VietNamPerson();
20         vietNam.say();
21
jb12.Welcome > main >
Search Results | Usages | Output - JB12 (run) ×
run:
Hello
Xin Chao
BUILD SUCCESSFUL (total time: 1 second)

```

- Một đối tượng thuộc kiểu Person khi triệu gọi phương thức say thì kết quả vẫn sẽ hiển thị ra chữ “Hello” đúng như nó đã định nghĩa phương thức này.
- Phương thức tĩnh (static) và Phương thức có access modifier là private không thể override được từ lớp con. Nếu lớp con có phương thức có cùng tên thì đó là phương thức hoàn toàn độc lập và phương thức của lớp cha sẽ bị ẩn đi.
- Chúng ta không thể thu hẹp Access modifier của một method kế thừa từ lớp cha. Nghĩa là nếu lớp cha đã khai báo public cho method thì khi override lại lớp con sẽ không thể thay đổi sang các access modifier có quyền truy cập nhỏ hơn như: protected, default hay private.

## 2.18 Lớp trừu tượng trong Java

- Khi ta thiết kế các lớp để dùng chung cho các lớp sau kế thừa và không để khai báo một đối tượng cụ thể nào cả, Lớp này cũng không biết phải thực thi các method cụ thể là gì thì chúng ta nghĩ ngay đến lớp trừu tượng (abstract).

Ví dụ Animal thì không có khai báo cụ thể là một thực thể nào cả vì mỗi loài đều có thể được khai báo bởi một class cụ thể hơn, chi tiết hơn. Animal có hành vi di chuyển thì mỗi một loài có cách cách di chuyển khác nhau rất đa dạng, vì vậy việc thực thi hành vi di chuyển này sẽ dành cho các lớp con kế thừa và override lại phương thức di chuyển này

- Các lớp trừu tượng được khai báo bằng từ khóa abstract

```

①  public abstract class Animal {
13    protected int high;
14    protected int weight;
15    protected int numberofFoot;
⑥  public abstract void move();
②  public abstract String say();
18 }

```

- Lớp trừu tượng chỉ dùng để lớp khác kế thừa và không thể được dùng như một kiểu dữ liệu để khai báo các biến đối tượng. Bởi vì các lớp class có thể có các method chưa được định nghĩa, nó cần phải được lớp con kế thừa và thực hiện override lại các method abstract này.
- Lớp trừu tượng được thiết kế để định nghĩa các thuộc tính và hành vi và nó không cần phải viết thân của method(viết phần xử lý của method) mà chỉ cần chỉ ra có các method gì, các method có tên, kiểu trả về và các tham số đầu vào.

```

①  public abstract class Animal {
13    protected int high;
14    protected int weight;
15    protected int numberofFoot;
⑥  public abstract void move();
②  public abstract String say();
18 }

```

- Lớp con nào nếu không phải là lớp trừu tượng mà kế thừa từ một lớp trừu tượng thì nó cần phải implement toàn bộ các method abstract của lớp trừu tượng đó.

```

①  public class Person extends Animal{
12    String name = "sophia";
13    protected int yearOfBirth;
14    protected boolean isMarried;
15    private static int nCounter; //Total of person.
16    private final int MAX_PERSON = 10000;
17
18    @Override
②    public void move() {
19      //Move by food
20    }
21
22    @Override
③    public String say() {
23      String text = "Hello";
24      System.out.println(text);
25      return text;
26    }
27

```

## 2.19 Interface trong Java

- Interface là một class đặc biệt, nó được dùng để định nghĩa các thuộc tính hằng và hành vi của các đối tượng, nhưng cũng giống như class abstract các hành vi này không cần định nghĩa phần body.

- Cài đặt một interface ta dùng từ khóa interface như sau:

```

12  public interface IExcecuteable {
13      public void execute();
14  }

```

- Lớp con có thể kế thừa hay nói chính xác ra là có thể thực thi (Implement) từ interface bằng từ khóa “implements”. Một khi đã implement từ interface thì class đó bắt buộc phải cài đặt body(implement) toàn bộ các method của interface. Nếu không thì sẽ phải là lớp abstract.

```

12  public class Bird implements IFlyAble{
13      @Override
14      public void fly() {
15          //Bay bằng cách vỗ cánh
16      }
17
18  }

```

Hoặc cài đặt implement nhiều interface như sau:

```

12  public class Bird implements IFlyAble, IAnimal{
13      @Override
14      public void fly() {
15          //Bay bằng cách vỗ cánh
16      }
17
18      @Override
19      public void move() {
20
21      }
22
23      @Override
24      public void eat() {
25
26  }
27
28  public interface IFlyAble {
29      public final static int MAX_SPEED = 1000; //km/h
30  }
31
32  public interface IAnimal {
33      public void move();
34      public void eat();
35  }

```

- Mọi phương thức trong interface đều là phương thức trừu tượng mặc dù không cần khai báo từ khóa abstract.

- Mọi thuộc tính trong interface đều là hằng (final) mặc dù ta không cần có từ khóa final nhưng nó vẫn là hằng, vì vậy ta cũng nên đặt làm biến static để tiết kiệm bộ nhớ. Trong lúc khai báo ta nên khai báo tường minh như sau:

```
13 |     public final static int MAX_SPEED = 1000; //km/h
```

- Mọi phương thức và thuộc tính đều chỉ có thể là public cho dù có từ khóa public hay không.
- Interface không có phương thức khởi động.
- Bản thân interface cũng là một thiết kế trừu tượng, vì vậy không thể có một đối tượng nào có kiểu interface được, biến đối tượng thì bình thường nhưng ta không thể cấp phát bộ nhớ bằng từ khóa new có kiểu interface được

```
22 |     IFlyAble fly;
23 |     fly = new IFlyAble();
```

Muốn có thể được khai báo instant thì chúng ta sẽ phải viết ngay một Anonymus class để thực thi các phương thức abstract.

```
22 |     IFlyAble fly;
23 |     fly = new IFlyAble() {
24 |         @Override
25 |         public void fly() {
26 |             //some code here
27 |         }
28 |     };
```

- Từ Java 1.8 trở đi Interface được phép định nghĩa phần body. Điều này nhằm mục đích tương thích giữa phần mã code cũ đã viết với các interface mới được update. Các interface mới khi được bổ sung thêm các method thì phần mã cũ chưa được implement các method này vẫn có thể biên dịch và chạy được bình thường vì nó sẽ sử dụng phần body của các method định nghĩa trong interface.

```
①  public interface IFlyAble {
13 |     public final static int MAX_SPEED = 1000; //km/h
14 |     public default void fly() {
15 |         //some code here
16 |     }
17 | }
```

- Interface được dùng với mục đích tạo ra một giao tiếp chung cho các đối tượng vì vậy ta nên vận dụng đúng ngữ cảnh thì tốt hơn.
- Bây giờ ta sẽ phân biệt khi nào thì dùng abstract class, khi nào thì dùng interface.
  - o Abstract class có xu hướng là bản thiết kế cho các đối tượng có cùng bản chất cả về thuộc tính lẫn method
  - o Interface có xu hướng là bản thiết kế cho các đối tượng có cùng hành vi nhưng không cần cùng bản chất.

Ta ví dụ như sau: loài chim và máy bay vốn không có cùng bản chất, như chúng lại có hoạt động bay, vì vậy được xếp vào những loại được định nghĩa là có thể bay được.

```

10  * @author tanth
11  */
12  public interface IFlyAble {
13      public void fly();
14  }

15
16  public class Bird implements IFlyAble{
17      @Override
18      public void fly() {
19          //Bay bằng cách vỗ cánh
20      }
21  }

22  public class Planes implements IFlyAble{
23
24      @Override
25      public void fly() {
26          //bay bằng cánh và động cơ
27      }
28  }

```

## 2.20 Tìm hiểu về tính chất hướng đối tượng (Object Oriented Programming) trong Java

- Hướng đối tượng thường sẽ có 4 tính chất, nếu một ngôn ngữ lập trình nào mà có đủ 4 tính chất sau thì sẽ gọi là ngôn ngữ lập trình hướng đối tượng. sau đây ta đi nghiên cứu từng tính chất một

### 2.20.1 Tính Trừu tượng (*abstraction*)

- Quá trình tìm hiểu và xem xét một đối tượng trong thực tế để phản ánh và mô tả được các đối tượng này vào trong chương trình. Trong khi thiết kế các đối tượng này có thể bỏ qua các thuộc tính và hành vi không quan trọng và không cơ bản để tập trung mô tả đối tượng với những thuộc tính và hành vi cơ bản và quan trọng. Quá trình này ta gọi là quá trình trừu tượng hóa đối tượng.
- Trong quá trình thiết kế các class thực chất đây là quá trình chúng ta đi trừu tượng hóa các đối tượng và sản phẩm của quá trình này chính là các class và object.
- Trong quá trình thiết kế thì đôi khi ta tạo lên các class mô tả các đối tượng không có thực trong thực tế và các class này thường được biết đến là các abstract class.

### 2.20.2 Tính đóng gói (*encapsulation*)

- Các thuộc tính và phương thức được chứa đựng hay nói cách khác là đóng gói vào một đơn vị đó là class. Class giúp ẩn giấu đi những gì không cần hiện diện đối với bên ngoài và chỉ cho thế giới bên ngoài sử dụng những gì nó cho phép. Sự phức tạp đã được che giấu đi đó là rất nhiều những thuộc tính và thuật toán xử lý ẩn chứa bên trong, điều này tạo nên tính trong sáng và dễ dùng khi ta sử dụng các lớp đối tượng.
- Các class cũng được đóng gói vào các package để trở thành những gói độc lập và tạo nên một gianh giới về truy cập.

### 2.20.3 Tính kế thừa (*inheritance*)trong Java

- Là khả năng thiết kế class được kế thừa lại class trước đó để kế thừa lại một số hoặc toàn bộ các thuộc tính và phương thức của lớp cha và xác lập quan hệ kế thừa với lớp cha.
- Tính kế thừa là một tính chất vô cùng quan trọng của lập trình hướng đối tượng. Tính kế thừa và tính đa hình là hai tính chất đi liền với nhau như thể là hai mặt của một đồng xu. Không có kế thừa thì cũng ko có đa hình
- Chúng xem xét kỹ hơn ở phần lớp kế thừa để tìm hiểu về cách thiết kế lớp kế thừa và các tính chất khi kế thừa.

### 2.20.4 Tính đa hình (*polymorphism*)

- Tính đa hình là khả năng của một đối tượng có thể được đại diện(trỏ tới) bởi các biến đối tượng khác nhau, các biến đối tượng này thuộc kiểu là chính nó hoặc là các kiểu của supper class trực

thuộc phả hệ mà nó kế thừa. Như vậy là một đối tượng có thể mang nhiều hình dạng khác nhau thì ta gọi là tính đa hình

- Tính đa hình để hiểu thì không hề khó, khó là có thể áp dụng được tính đa hình vào trong bài toán của chúng ta, rất nhiều các design pattern đã hình thành dựa trên tính đa hình này. Tính đa hình làm cho bài toán trở lên vô cùng mềm dẻo nhưng sẽ khó hiểu, khó bảo trì mã code hơn vì chúng ta sẽ khó theo dõi luồng xử lý nghiệp vụ vì không biết thực chất hành vi của method thực sự nằm ở đối tượng nào được thực hiện. Sau đây chúng ta sẽ đi nghiên cứu ứng dụng của đa hình vào trong trường hợp cụ thể

- o Ta có một interface IFlyAble

```

12  public interface IFlyAble {
13      public final static int MAX_SPEED = 1000; //km/h
14      public default void fly() {
15          System.out.println("I can fly");
16      }
17  }

```

- o Khai báo tiếp một class Bird được implement từ IFlyAble

```

12  public class Bird implements IFlyAble, IAnimal{
13      @Override
14      public void fly() {
15          System.out.println("I am a bird, I fly with my wings");
16      }

```

- o Và class Planes được implement từ IFlyAble

```

12  public class Planes implements IFlyAble{
13      @Override
14      public void fly() {
15          System.out.println("I'm an airplane, I can fly with wings and engines");
16      }
17
18  }

```

- o Vậy giờ ta khai báo và thi triển tính đa hình của đối tượng bird và planes như sau:

```

22     Bird bird1;
23
24     Planes planes1;
25     IFlyAble fly1;
26
27     bird1 = new Bird();
28     planes1 = new Planes();
29
30     bird1.fly();
31     planes1.fly();
32
33     fly1 = bird1;
34     fly1.fly();
35
36     fly1 = planes1;
37     fly1.fly();

```

Kết quả như sau:

```

run:
I am a bird, I fly with my wings
I'm an airplane, I can fly with wings and engines
I am a bird, I fly with my wings
I'm an airplane, I can fly with wings and engines
BUILD SUCCESSFUL (total time: 8 seconds)

```

- Như vậy từ ví dụ trên ta thấy cùng là đối tượng bird nhưng được 2 biến đối tượng là bird1 (dòng 27) và fly1 (dòng 33) tham chiếu tới. Hai đối tượng này thuộc hai kiểu khác nhau là Bird và IFlyAble. Và khi thực hiện cùng một phương thức fly() thì đều là một đối tượng thực hiện mà thôi nên kết quả thực hiện là giống nhau.
- Chúng ta để ý thấy biến đối tượng fly1 có thể tham chiếu tới đối tượng của biến bird1 và plaines1 bởi vì hai biến đối tượng này có kiểu dữ liệu là class đã kế thừa từ IFlyAble.
- Chúng ta tiếp tục xét thêm một trường hợp nữa của ví dụ này như sau:
  - Một phương thức **whoCanFly (IFlyAble flyObject)** được triệu gọi, nhưng điều đặc biệt là phương thức này có tham số đầu vào là Interface **IFlyAble flyObject**. Vậy là tất cả các biến đối tượng cứ kế thừa từ IFlyAble là đều có thể được truyền vào method này, điều này cho thấy bất cứ đối tượng nào kế thừa từ IFlyAble khi truyền vào method **whoCanFly** thì trong method chúng đều mang hình dạng của một biến đối tượng IFlyAble. Và trong Method này chúng ta không cần quan tâm tới việc đối tượng nào thực sự được truyền vào và chúng ta chỉ việc dùng đối tượng đó với những method đã khai báo ở interface IFlyAble.

```

21  public static void main(String[] args) {
22
23      Bird birdl;
24      Planes planesl;
25      IFlyAble flyl;
26
27      birdl = new Bird();
28      planesl = new Planes();
29
30      birdl.fly();
31      planesl.fly();
32      System.out.println("1 -----");
33      flyl = birdl;
34      flyl.fly();
35      System.out.println("2 -----");
36      flyl = planesl;
37      flyl.fly();
38      System.out.println("3 -----");
39      whoCanFly(birdl);
40      whoCanFly(planesl);
41  }
42  public static void whoCanFly(IFlyAble flyObject) {
43      flyObject.fly();
44  }

```

- o Và đối tượng nào được truyền vào method thì nó sẽ thực hiện hành vi của đúng đối tượng đó. kết quả như sau:

```

run:
I am a bird, I fly with my wings
I'm an airplane, I can fly with wings and engines
1 -----
I am a bird, I fly with my wings
2 -----
I'm an airplane, I can fly with wings and engines
3 -----
I am a bird, I fly with my wings
I'm an airplane, I can fly with wings and engines
BUILD SUCCESSFUL (total time: 1 second)

```

## 2.21 Chú ý trong truyền đối số vào method trong Java (Passing Data Type Arguments)

- Các method cho phép truyền đối số (Arguments) vào để nó thực hiện các giải thuật logic và trả ra kết quả. Trong quá trình đó các đối số được truyền vào và method và trở thành các tham số (Parameters) trong Method và tham gia vào các giải thuật logic. Các parameter này có thể bị thay đổi giá trị trong quá trình tính toán của giải thuật trong method.

- Điều chúng ta quan tâm là sau khi thực hiện xong method thì các Arguments này có bị thay đổi giá trị hay không, câu trả lời là các Argument hoàn toàn không bị thay đổi giá trị trong mọi trường hợp và có chăng chỉ là đối tượng mà nó trả đến có khả năng bị thay đổi giá trị trong trường hợp đối số là biến đối tượng (Biến reference).
- Sau đây ta đi nghiên cứu đến 2 trường hợp truyền đối số là biến Primitive và biến Reference để biết tại sao lại như vậy nhé.

### 2.21.1 Truyền đối số Primitive

- Chúng ta đi xét ví dụ sau:
  - o hai biến số a và b là tham số của method add(int a, int b), và tham số a bị thay đổi giá trị trong quá trình tính toán ở bên trong của method add.

```

17   public static int add(int a, int b) {
18     a = a + b;
19     return a;
}

```

- o Hai biến số x và y là các đối số của method add(int a, int b);

```

21   int x = 5;
22   int y = 6;
23   int z = 0;
24   z = add( x, y );
25   System.out.println("x = " + x);
26   System.out.println("y = " + y);
27   System.out.println("x = " + z);

```

Vậy sau khi gọi xong method add ở dòng 24 thì giá trị của x có bị thay đổi không ? và đây là câu trả lời.

```

x = 5
y = 6
z = 11
BUILD SUCCESSFUL (total time: 1 second)

```

- Vậy kết luận là: sau khi ra khỏi lời gọi method thì các đối số **Primitive** không bị thay đổi giá trị.

### 2.21.2 Truyền đối số Reference

- Ta xét tới một ví dụ giả định như sau: method rename có nhiệm vụ đặt lại tên cho đối tượng person với name được truyền vào từ đối số newName tại dòng 21.

- Tại dòng 22 đổi đối tượng person được khởi tạo lại đổi đối tượng khác và được đặt tên mới tại dòng 23.

```

20 |   public static String rename(Person person, String newName) {
21 |     person.setName(newName);
22 |     person = new Person();
23 |     person.setName("Nguyen Hoang Manh Dung");
24 |     return person.getName();
25 |

```

- Ta khởi tạo đối tượng person1 và thực hiện đổi tên bằng method **rename**.

```

26 |   public static void main(String[] args) {
27 |     Person person1 = new Person();
28 |     person1.setName("Nguyen Van Nam");
29 |     System.out.println("My older name is " + person1.getName());
30 |     String name = rename(person1, "Nguyen Thi Mo");
31 |     System.out.println("My new name is " + name);
32 |     System.out.println("My Real new name is " + person1.getName());
33 |

```

- Chúng ta kiểm chứng các kết quả thu được như sau:

run:  
 My older name is Nguyen Van Nam  
 My new name is Nguyen Hoang Manh Dung  
 My Real new name is Nguyen Thi Mo  
 BUILD SUCCESSFUL (total time: 1 second)

Vậy kết luận là:

- Đối số là biến đổi đối tượng thì không bị thay đổi giá trị khi ra khỏi method (dòng 22 và 23 tại ví dụ).  
cụ thể là khi ra khỏi method rename biến đổi đối tượng person1 vẫn trả về đối tượng đã khởi tạo bên ngoài tại dòng 27 mặc dù bên trong method rename đã bị trả về một đối tượng khác bởi dòng 22
- Đối tượng thì có thể bị thay đổi ở bất cứ chỗ nào (dòng 21 tại ví dụ). Cụ thể là đối tượng bên ngoài được trả bởi biến đổi đối tượng person được thay đổi giá trị name tại dòng 21 và khi ra khỏi method rename đối tượng này đã thực sự bị thay đổi giá trị.

## 2.22 Kiểu dữ liệu bao bọc (Wrapper classes)

- Trong Java với mỗi kiểu dữ liệu nguyên thủy (primitive type) thì đều có một lớp (class) dữ liệu khác tương ứng bao bọc. Ví dụ kiểu dữ liệu Integer chứa đựng bên trong nó một giá trị int.

- Kiểu dữ liệu wrapper như một cái túi bao bọc lấy nhân của nó là kiểu dữ liệu nguyên thủy tương ứng và còn cung cấp thêm các methods để dễ dàng thao tác với dữ liệu nguyên thủy mà nó chưa đựng, chính vì vậy mà ta gọi là kiểu dữ liệu bao bọc (wrapper).
- Kiểu wrapper được sử dụng khi chúng ta muốn làm việc với các kiểu dữ liệu dạng Object và cần sự hỗ trợ của các method với các thuật toán có sẵn của kiểu wrapper.
- Sau đây là các kiểu dữ liệu wrapper:

Primitive Data type	Wrapper class	Constructor Arguments
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	double float or String
double	Double	double or String

- Do là các đối tượng nên khi ta "khai báo kiểu Wrapper" thì giá trị mặc định của biến đối tượng là null, còn đối với kiểu "nguyên thuỷ" thì giá trị mặc định của biến được gán là 0. Đây là sự khác biệt cơ bản, vì vậy trước khi sử dụng các biến đối tượng wrapper chúng ta phải nhất thiết khởi tạo đối tượng và giá trị của nó thông qua một trong các cách sau:

```

27 |     Integer numberOne = new Integer("1");
28 |     System.out.println("The first Value of numberOne is " + numberOne);
29 |     Delay snip creation
30 |     numberOne = new Integer(2);
31 |     System.out.println("The second Value of numberOne is " + numberOne);
32 |     numberOne = 3;
33 |     System.out.println("The third Value of numberOne is " + numberOne);
34 |     numberOne = Integer.valueOf("4");
35 |     System.out.println("The 4th Value of numberOne is " + numberOne);
36 |     numberOne = Integer.parseInt("5");
37 |     System.out.println("The 5th Value of numberOne is " + numberOne);

```

- Chú ý dòng số 29, chúng ta không cần thiết phải khai báo với một từ khóa new vì java sẽ tự chuyển đổi (autoboxing) giá trị int sang kiểu của biến đối tượng Integer và vì vậy chúng ta có thể dùng một cách tiện lợi như tại dòng số 31.
- Và sau đây là kết quả thu được



```

The first Value of numberOne is 1
The second Value of numberOne is 2
The third Value of numberOne is 3
The 4th Value of numberOne is 4
The 5th Value of numberOne is 5
BUILD SUCCESSFUL (total time: 1 second)

```

- Sau đây là mô tả về 2 wrapper Integer và Double

java.lang.Integer	java.lang.Double
<u>-value: int</u> <u>+MAX_VALUE: int</u> <u>+MIN_VALUE: int</u>  <u>+Integer(value: int)</u> <u>+Integer(s: String)</u> <u>+byteValue(): byte</u> <u>+shortValue(): short</u> <u>+intValue(): int</u> <u>+longValue(): long</u> <u>+floatValue(): float</u> <u>+doubleValue(): double</u> <u>+compareTo(o: Integer): int</u> <u>+toString(): String</u> <u>+valueOf(s: String): Integer</u> <u>+valueOf(s: String, radix: int): Integer</u> <u>+parseInt(s: String): int</u> <u>+parseInt(s: String, radix: int): int</u>	<u>-value: double</u> <u>+MAX_VALUE: double</u> <u>+MIN_VALUE: double</u>  <u>+Double(value: double)</u> <u>+Double(s: String)</u> <u>+byteValue(): byte</u> <u>+shortValue(): short</u> <u>+intValue(): int</u> <u>+longValue(): long</u> <u>+floatValue(): float</u> <u>+doubleValue(): double</u> <u>+compareTo(o: Double): int</u> <u>+toString(): String</u> <u>+valueOf(s: String): Double</u> <u>+valueOf(s: String, radix: int): Double</u> <u>+parseDouble(s: String): double</u> <u>+parseDouble(s: String, radix: int): double</u>

- Các method tĩnh hữu ích trong việc phân tích và chuyển đổi dữ liệu từ một String hoặc wrapper sang giá trị wrapper hoặc primitive và ngược lại. Ví dụ như phương thức dưới đây

```
Integer.parseInt(String s)
```

```
Integer.parseInt(String s, int radix)
```

- o Trong đó radix được hiểu là cơ số của số cần chuyển đổi, đó là các cơ số 2,8,10,16..., nếu  
Nếu ta không chỉ ra radix thì cơ số mặc định là 10

```
37     numberOne = Integer.parseInt("11",10);
38     System.out.println("The 6th Value of numberOne is " + numberOne);
39     numberOne = Integer.parseInt("B",16);
40     System.out.println("The 7th Value of numberOne is " + numberOne);
```

- o Cơ số là hệ số đếm, ta có các hệ số đếm phổ biến như nhị phân, bát phân, thập phân và  
thập lục phân. Trong đời sống thường ngày chúng ta quen dùng cơ số 10 là thập phân,  
trong tính toán mặc định của Java cũng dùng hệ số đếm mặc định là thập phân, tuy nhiên  
các cơ số đếm khác vẫn được sử dụng rộng rãi trong tính toán và hỗ trợ hoàn toàn bởi  
Java và hầu hết các ngôn ngữ lập trình khác.
- o Trong ví dụ trên tại dòng 37 là cơ số 10 và tại dòng 39 là cơ số 16
- o Cùng là một giá trị String đưa vào là “11” thì tại dòng 37 có cơ số 10 và dòng 43 dùng cơ số  
2 lại cho ra 2 giá trị khác nhau

```
43     numberOne = Integer.parseInt("11",2);
44     System.out.println("The 9th Value of numberOne is " + numberOne);
45
```

Giá trị đạt được của numberOne như sau:

```
The first Value of numberOne is 1
The second Value of numberOne is 2
The third Value of numberOne is 3
The 4th Value of numberOne is 4
The 5th Value of numberOne is 5
The 6th Value of numberOne is 11
The 7th Value of numberOne is 11
The 9th Value of numberOne is 3
BUILD SUCCESSFUL (total time: 1 second)
```

- Khi đã dùng cơ số thì chúng ta cần chỉ ra chính xác cơ số của giá trị đưa vào như ví dụ dòng 39, nếu không chính xác chúng ta sẽ nhận về một lỗi ngoại lệ NumberFormatException

```

37     numberOne = Integer.parseInt("11",10);
38     System.out.println("The 6th Value of numberOne is " + numberOne);
39     numberOne = Integer.parseInt("B",16);
40     System.out.println("The 7th Value of numberOne is " + numberOne);
41     numberOne = Integer.parseInt("B",10);
42     System.out.println("The 8th Value of numberOne is " + numberOne);
43

```

Search Results   Usages   Output - JB12 (run) ×   Git Repository Browser

```

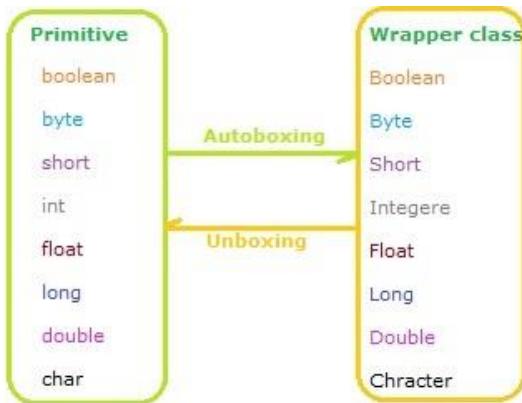
run:
The first Value of numberOne is 1
The second Value of numberOne is 2
The third Value of numberOne is 3
The 4th Value of numberOne is 4
Exception in thread "main" java.lang.NumberFormatException: For input string: "B"
The 5th Value of numberOne is 5
The 6th Value of numberOne is 11
The 7th Value of numberOne is 11
at java.lang.NumberFormatException.forInputString (NumberFormatException.java:65)
at java.lang.Integer.parseInt (Integer.java:580)
at jbl2.Welcome.main (Welcome.java:41)

```

như sau:

## 2.23 Autoboxing and Unboxing trong Java

- Khi nghiên cứu về kiểu dữ liệu nguyên thủy chúng ta đã tìm hiểu về chuyển đổi giữa các kiểu dữ liệu và quá trình ép kiểu (casting). Và bây giờ chúng ta xem xét đến một khái niệm tương tự nhưng là đối với kiểu dữ liệu wrapper, đó là quá trình boxing
- Quá trình chuyển đổi từ kiểu dữ liệu nguyên thủy sang kiểu dữ liệu wrapper tương ứng được gọi là quá trình Autoboxing hay còn gọi là quá trình tự đóng gói, ngược lại từ wrapper sang kiểu dữ liệu nguyên thủy gọi là Unboxing hay còn gọi là quá trình mở gói.



- Sau đây là ví dụ về quá trình Autoboxing tại dòng 28, từ kiểu dữ liệu int được tự động chuyển đổi sang kiểu Integer.

```

28 |     Integer numberOne = 3;
29 |     System.out.println("The third Value of numberOne is " + numberOne);

```

- Sau đây là ví dụ về quá trình Unboxing tại dòng 30 khi java tự chuyển đổi kiểu dữ liệu Integer về int.

```

28 |     Integer numberOne = 3;
29 |     System.out.println("The third Value of numberOne is " + numberOne);
30 |     int x = numberOne;
31 |     System.out.println("Unboxing numberOne to int x = " + x );

```

Và kết quả là:

```
The third Value of numberOne is 3
Unboxing numberOne to int x = 3
```

## 2.24 Kiểu dữ liệu String

- String là một kiểu dữ liệu quan trọng trong Java. Để biểu diễn dữ liệu như tên, ghi chú, mô tả... ta sử dụng String thay vì một mảng char hoặc mảng Character.
- String trong Java hỗ trợ Unicode 16bit vì vậy có thể biểu diễn(chứa đựng) được tiếng việt tiếng trung và nhiều loại ngôn ngữ khác trên thế giới.
- Chúng ta có 3 cách khởi tạo một đối tượng String như sau:

```

15 |     //cách 1
16 |     String myName1 = new String("Active Study");
17 |     //cách 2
18 |     String myName2 = "Active Study";
19 |     //cách 3
20 |     char[] name = {'A','c','t','i','v','e',' ', 's','t','u','d','y'};
21 |     String myName3 = new String(name);

```

- Ta có thể tiến hành cộng chuỗi với nhau hoặc cộng chuỗi với các biến số khác để tạo ra chuỗi mới

```

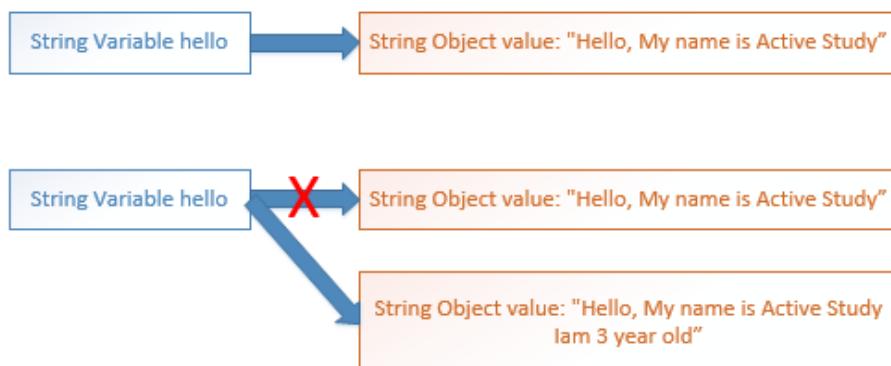
23 |     String hello = "Hello, My name is " + myName1;
24 |     int myAge = 3;
25 |     hello = hello + " I am " + myAge + " year old.";
26 |     System.out.println(hello);

```

Và kết quả là:

```
run:
Hello, My name is Active Study Iam 3 year old.
BUILD SUCCESSFUL (total time: 1 second)
```

- Một đối tượng String trong java là không thể thay đổi (immutable), thế điều quái gì đã xảy ra ở ví dụ trên khi mà ta đi cộng chuỗi cho biến hello, rõ ràng biến hello đã thay đổi giá trị cơ mà. Điều này đúng là biến hello đã thay đổi giá trị nhưng là do nó trỏ tới một đối tượng khác vừa được tạo ra từ phép cộng chuỗi ở dòng số 25.



- Vậy là mỗi lần thay đổi giá trị của đối tượng là một đối tượng mới lại được tạo ra cho dù là thay đổi nhỏ nhất, điều này thực sự có làm cho JVM phải hao tổn bộ nhớ không khi mà có rất rất nhiều thao tác thay đổi giá trị biến String trong một chương trình Java. Các bạn đừng lo lắng, để giải quyết điều này Java có 2 giải pháp cho chúng ta.
  - o Một là ta có thể dùng biến String khác để thay thế đó là **StringBuilder** hoặc **StringBuffer** để khai báo đối tượng String có nhiều thay đổi
  - o Hai là Java tổ chức “**String Pool**” để lưu trữ các giá trị String và dùng lại cho những lần sau khi khai báo ra các biến String có cùng giá trị. Chúng ta đi nghiên cứu ví dụ về String Pool thông qua ví dụ về khai báo biến myName2 như sau:

```
18 |     String myName2 = "Active Study";
```

khi chạy dòng 18 thì biến myName2 được tạo ra và JVM sẽ tìm trong “String pool” xem có chuỗi nào tương tự có giá trị “Active Study” đã được tạo ra từ trước đó chưa?

- Nếu chưa có thì một đối tượng chuỗi mới được tạo ra và được đưa vào “String pool”, và biến đối tượng myName2 sẽ tham chiếu tới đó.
  - Nếu đã có rồi thì JVM sẽ không tạo ra một đối tượng chuỗi mới nữa để tránh bị trùng lặp và biến đối tượng myName2 sẽ trỏ tới chuỗi đã được tạo ra trước đó có giá trị “Active Study” đang nằm ở trong String pool, điều này làm giảm thiểu chuỗi mới được tạo ra nên đỡ tốn bộ nhớ.
- Trong 3 cách khai báo String ở trên thì ta nên sử dụng cách khai báo theo cách nào, đương nhiên là ta đang phân vân giữa cách 1 và cách 2 rồi

```

17   String myName0 = "Active Study";
18
19   String myName1 = new String("Active Study");
20
21   String myName2 = "Active Study";
22
23   System.out.println("myName1 == myName2 is " + (myName1 == myName2));
24   System.out.println("myName2 == myName0 is " + (myName2 == myName0));

```

- Chúng ta sẽ đi phân tích về sự khác nhau của cách thức tổ chức String của 2 cách trong bộ nhớ để biết nên sử dụng cách nào trong 2 cách trên nhé.
  - Đối với cách 1 thì biến đối tượng myName1 sẽ được tạo ra và một đối tượng String sẽ được tạo ra ở trong bộ nhớ Heap mặc dù trong String pool đã tồn tại một đối tượng có giá trị tương tự. Và đối tượng mới được tạo ra này sẽ không được đưa vào String Pool để quản lý.
  - Đối với cách 2 thì biến đối tượng myName2 sẽ cũng được tạo ra và một đối tượng String đã được tạo ra ở trong String Pool được biến myName2 trỏ tới. và kết quả của việc so sánh hai đối tượng myName2 và myName0 thể hiện được điều này:

```

myName1 == myName2 is false
myName2 == myName0 is true
BUILD SUCCESSFUL (total time: 0 seconds)

```

- String được implement từ các interface Serializable, Comparable và CharSequence. Điều này thể hiện như sau:
  - String sẽ có thể được ghi đầy đủ xuống một file hoặc một Stream nào đó để sau đó được deSerializable trả lại mà vẫn vẹn nguyên giá trị của các thuộc tính bên trong.

- o String có thêm phương thức về so sánh giữa hai string
- o String có phương thức thao tác với chuỗi giá trị char mà nó nắm giữ như : length(), charAt(), subsequence()...

```

111  public final class String
112      implements java.io.Serializable, Comparable<String>, CharSequence {
113  ☐      /** The value is used for character storage. */
114      private final char value[];

```

#### 2.24.1 Replacing and Splitting Strings

- `public String replace(char oldChar, char newChar)`

method này tạo ra một chuỗi mới đã được thay tất cả các ký tự **oldChar** thành ký tự **newChar**

- `public String replaceAll(String regex, String replacement)`

method này tạo ra một chuỗi mới đã được thay thế toàn bộ những đoạn string (substring) bên trong giống regular expression **regex** bằng đoạn string **replacement**

- `public String replaceFirst(String regex, String replacement)`

method này tạo ra một chuỗi mới đã được thay thế đoạn string (substring) đầu tiên bên trong giống regular expression **regex** bằng đoạn string **replacement**

- `public String replace(CharSequence target, CharSequence replacement)`

```

25      String welcome1 = "Welcome".replace('e', 'A');
26      String welcome2 = "Welcome".replaceFirst("e", "AB");
27      String welcome3 = "Welcome".replace("e", "AB");
28      String welcome4 = "Welcome".replace("el", "AB");
29
30      System.out.println(welcome1);
31      System.out.println(welcome2);
32      System.out.println(welcome3);
33      System.out.println(welcome4);

run:
WAcomA
WABlcome
WABlcomAB
WABcome

```

- `public String[] split(String regex)`

Method chia nhỏ String thành các chuỗi nhỏ hơn chứa đựng trong một mảng String mới, dấu hiệu để tách thành các chuỗi nhỏ là do chúng được ngăn cách nhau bởi chuỗi **regex** ở trong chuỗi bị chia nhỏ. Xét ví dụ chuỗi sau được chia nhỏ bởi ký tự ngăn cách “,”.

```

42     String lstName = "Nguyen Thi Mo,Nguyen Duc Tan,Khuat Dang Quang";
43     String[] arrName = lstName.split(",");
44     for (String name : arrName) {
45         System.out.println(name);
46     }

```

Và Kết quả là:

```

run:
Nguyen Thi Mo
Nguyen Duc Tan
Khuat Dang Quang
BUILD SUCCESSFUL (total time: 0 seconds)

```

- `public String[] split(String regex, int limit)`  
cũng giống như `split` nhưng method này giới hạn số phần tử bị chia đầu ra bởi tham số `limit`

#### 2.24.2 Matching, Replacing and Splitting by Patterns

- Class `String` cung cấp khả năng tìm kiếm, thay thế và chia chuỗi theo một khuôn mẫu (Pattern). Pattern này có cú pháp và chúng ta gọi là “regular expression” và chúng ta sẽ tìm hiểu một chút về cú pháp này thông qua các ví dụ như sau:

```

35     String welcome5 = "Welcome to Active Study";
36     boolean isMatch = welcome5.matches("Welcome.*");
37     System.out.println("1. Match is " + isMatch);
38     isMatch = "Contact 08-98-98-2296".matches(".*.\d{2}-\d{2}-\d{2}-\d{4}");
39     System.out.println("2. Match is " + isMatch);

run:
1. Match is true
2. Match is true
BUILD SUCCESSFUL (total time: 0 seconds)

```

- o Dòng số 36 kiểm tra String `welcome5` có tồn tại chuỗi con “`Welcome`” ở đầu và tiếp theo là một hoặc nhiều ký tự nữa. và kết quả trả về `true`;
- o Dòng số 38 kiểm tra String có dạng như pattern “`.*.\d{2}-\d{2}-\d{2}-\d{4}`” hay không, và kết quả trả về là `true`. Ý nghĩa của pattern đó như sau:
  - “`*`” biểu thị chuỗi là một hoặc nhiều ký tự, và nó nằm ở đầu thì biểu thị là một hoặc nhiều ký tự ở vị trí đầu, ứng với giá trị “`Contact`”.

- \d{2} biểu thị hai số liền nhau
  - \d{4} biểu thị 4 số liền nhau.
- Ví dụ tiếp theo về ReplaceAll

```
40 |     welcome5 = "Contact 08-98-98-2296".replaceAll("\d{4}", "xxx");
41 |     System.out.println(welcome5);
```

```
Contact 08-98-98-xxx
BUILD SUCCESSFUL (total time: 0 seconds)
```

Chúng ta thấy ký tự 2296 đã được thay thế bởi “xxx” để che dấu đi 4 số cuối của số đt.

- Xét thêm một ví dụ về split để tách tên thành các tên riêng biệt nhờ vào dấu hiệu là các số đứng đầu mỗi tên, bây giờ chúng ta sẽ dùng expression sau: “\d{1}.”

```
42 |     String lstName = "1. Nguyen Thi Mo 2. Nguyen Duc Tan 3. Khuat Dang Quang";
43 |     String[] arrName = lstName.split("\d{1}.");
44 |     for (String name : arrName) {
45 |         System.out.println(name);
46 |     }
```

Và kết quả là:

```
Nguyen Thi Mo
Nguyen Duc Tan
Khuat Dang Quang
BUILD SUCCESSFUL (total time: 0 seconds)
```

### 2.24.3 Conversion between Strings and Arrays

- Chuyển đổi một chuỗi sang một array char.

```
47 |     String welcome5 = "Welcome to Active Study";
48 |     char[] arrChar = welcome5.toCharArray();
49 |     for (char CharE : arrChar) {
50 |         System.out.println(CharE);
51 |     }
```

Và kết quả là:

```

run:
W
e
l
c
o
m
e

t
o

A
c
t
i
v
e

s
t
u
d
y
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Để chuyển đổi ngược lại ta khởi tạo một String mới và truyền char[] vào phương thức khởi dụng hoặc phương thức valueOf()

```

47 |     String welcome5 = "Welcome to Active Study";
48 |     char[] arrChar = welcome5.toCharArray();
49 |     for (char CharE : arrChar) {
50 |         System.out.println(CharE);
51 |     }
52 |     String welcom6 = new String(arrChar);
53 |     System.out.println(welcom6);

```

Và kết quả là:

```

Welcome to Active Study
BUILD SUCCESSFUL (total time: 0 seconds)

```

#### 2.24.4 Formatting Strings

- Đôi khi ta muốn String có thể được định dạng theo một hình thức mà chúng ta mong muốn và để làm được điều này lớp String cung cấp một static method và cú pháp sau:
  - o **String.format(format, item1, item2, ..., itemk)**
  - o Trong đó **format** là định dạng theo cú pháp để hiển thị các item theo một định dạng mong muốn.

```

55 |     String s = String.format("%7.2f%6d%-4s", 45.556, 14, "AB");
    |     System.out.println(s);

```

- o “%7.2f” : biểu thị dành 7 vị trí, trong đó có 2 vị trí cho sau dấu phẩy động và f biểu thị đây là kiểu **float** đối với số ở vị trí đầu tiên “45.556”.
- o “%6d” : biểu thị dành sáu vị trí để biểu diễn **số nguyên** “14”
- o “%-4s” : Biểu thị dành 4 vị trí để biểu diễn chuỗi “AB” và “AB” được đưa lên đầu.

Kết quả như sau:

```

run:
45.56 14AB
BUILD SUCCESSFUL (total time: 0 seconds)

```

□□45.56□□□14AB□□

- Chú ý mỗi một vị trí được bắt đầu bởi % thì cần phải có đối số tương ứng.

## 2.25 Kiểu dữ liệu **StringBuilder** và **StringBuffer**

- Để giải quyết vấn đề chuỗi ký tự cần phải thay đổi liên tục thì Java cung cấp 2 class **StringBuilder** và **StringBuffer**. Hai lớp này sử dụng gần tương tự như lớp **String** nhưng có điểm khác biệt đó là có thể thay đổi được giá trị của Object mà không cần phải tạo Object khác và hai lớp này được trang bị thêm các phương thức để Insert thêm String hoặc ký tự vào một vị trí bất kỳ...
- **StringBuilder** và **StringBuffer** cũng giống nhau, chỉ khác là **StringBuffer** được sinh ra để làm việc trong chế độ multithread (đa tiến trình), bởi vì các method của **StringBuffer** có synchronized. Chúng ta sẽ tìm hiểu về Multithread và đồng bộ tiến trình ở các chương sau.
- Để tạo ra một đối tượng **StringBuilder** hoặc **StringBuffer** ta phải khởi tạo đối tượng

```

58 |     StringBuilder strContent = new StringBuilder();
59 |     strContent = new StringBuilder("Welcome to Active Study");
60 |     strContent = new StringBuilder(1000);
61 |
62 |
63 |     StringBuffer strContentBuff = new StringBuffer();
64 |     strContentBuff = new StringBuffer("Welcome to Active Study");
65 |     strContentBuff = new StringBuffer(1000);
66 |

```

- Ví dụ về thêm một nội dung vào cuối hoặc vào một vị trí bất kỳ trong chuỗi.

```

60      StringBuilder strContent = new StringBuilder("Welcome to Active Study");
61      System.out.println(strContent);
62      strContent.append(" we are at the Tran Vy");
63      System.out.println(strContent);
64      strContent.insert(strContent.indexOf("Tran Vy"), "52 ");
65      System.out.println(strContent);

```

- Và kết quả:

```

run:
Welcome to Active Study
Welcome to Active Study we are at the Tran Vy
Welcome to Active Study we are at the 52 Tran Vy
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Đã biết về String, StringBuilder và StringBuffer vậy chúng ta nên sử dụng kiểu dữ liệu nào cho chương trình của chúng ta.
  - o So sánh về tốc độ xử lý thì StringBuilder là nhanh nhất sau đó đến String và StringBuffer là chậm nhất vì StringBuffer phải xử lý đồng bộ (synchrolized)
  - o So sánh về độ an toàn đối với chương trình có multithread thì StringBuffer là an toàn tránh xung đột sau đó đến StringBuilder và cuối cùng là String. Bởi String không những không đồng bộ giống StringBuilder mà một đối tượng String còn được nhiều biến đổi tương cùng lúc tham chiếu tới.
  - o Nếu chúng ta có xu hướng sử dụng một chuỗi cố định, sử dụng nhiều lần và không thay đổi nội dung quá nhiều thì ưu tiên sử dụng String.

## 2.26 Tạo, biên dịch và chạy ứng dụng Java đơn giản trên môi trường window mà không cần IDE

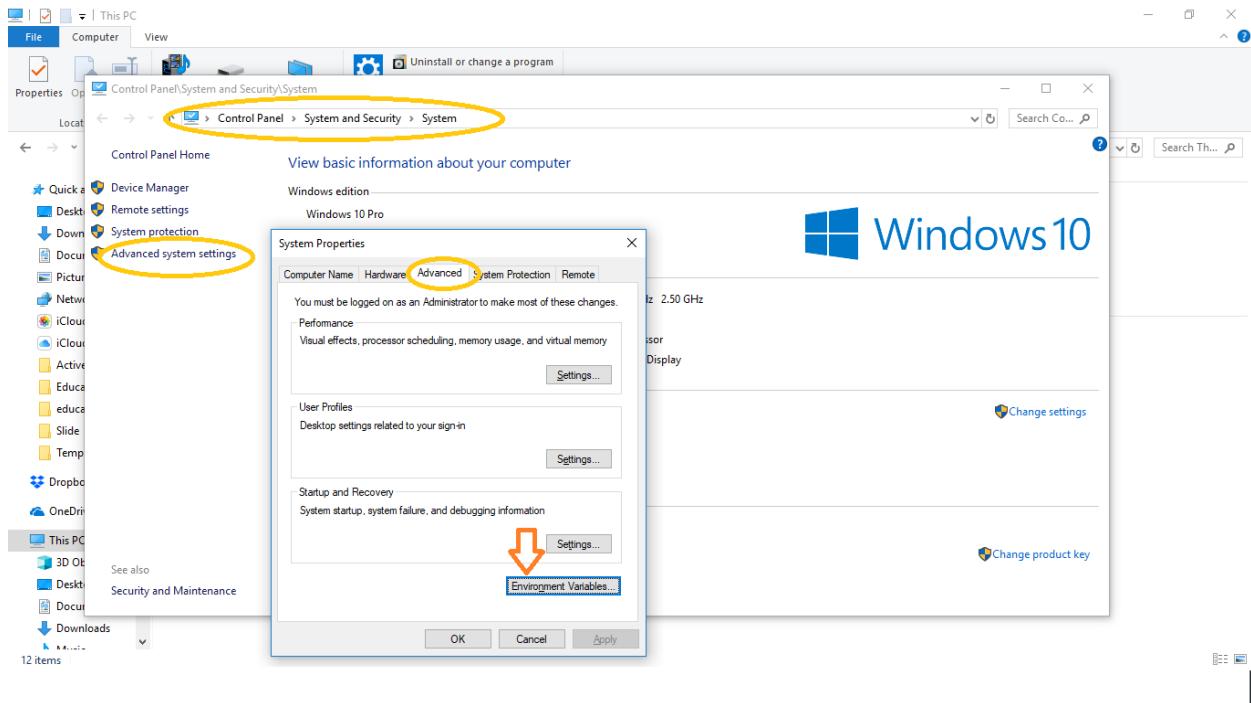
### 2.26.1 Cài đặt biến môi trường

#### ❖ Cài đặt biến môi trường trên môi trường window

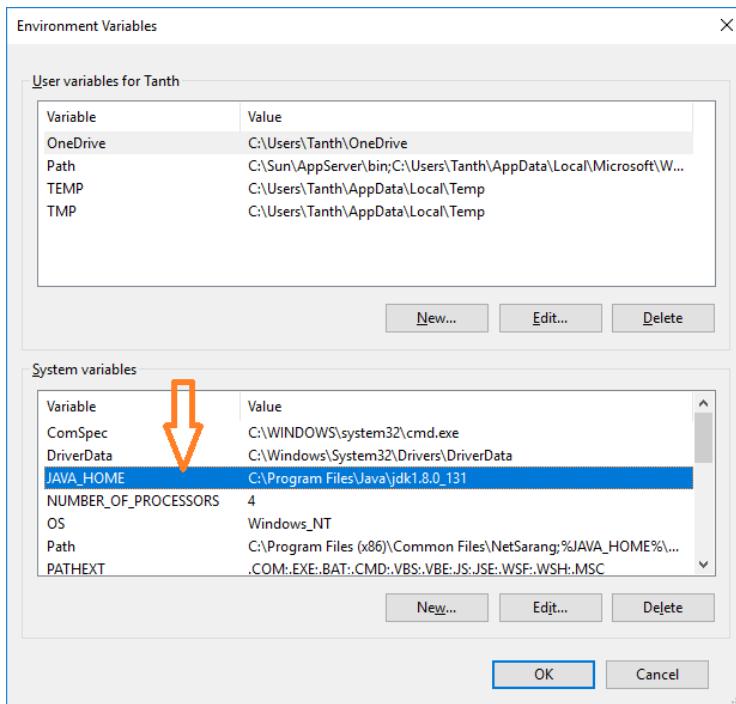
- Tại sao chúng ta phải cài đặt biến môi trường? Bởi vì các chương trình Java muốn được khởi chạy thì lập trình viên thậm chí cả người dùng cần phải gọi “Java.exe” để khởi chạy được máy ảo JVM và truyền vào cho JVM biết file mã nguồn cần chạy.
- Cũng giống như khi khởi chạy thì quá trình biên dịch cũng vậy cần chỉ ra “javac.exe”... nằm ở đâu.
- Các hệ điều hành khác nhau đều có khái niệm biến môi trường và các cách thiết lập biến môi trường là khác nhau, nhưng nguyên tắc là giống nhau, ví dụ như Window thì hệ điều hành sử dụng biến “Path” là biến mặc định. Khi cần tìm tới một tài nguyên file, thư viện nào đó thì hệ điều hành sẽ tìm kiếm trong biến “Path”. Tuy nhiên chúng ta có thể tạo ra những biến môi trường riêng của chúng ta, nhưng khi cần sử dụng thì chúng ta cần gọi tên biến

môi trường này ra. Ở trường hợp này chúng ta sử dụng biến môi trường “Path” làm mặc định và biến môi trường khác nữa đó là “JAVA\_HOME” do chúng ta tạo ra để làm tường minh và giúp biến “Path” được gọn gàng hơn. Sau đây là các bước làm cụ thể

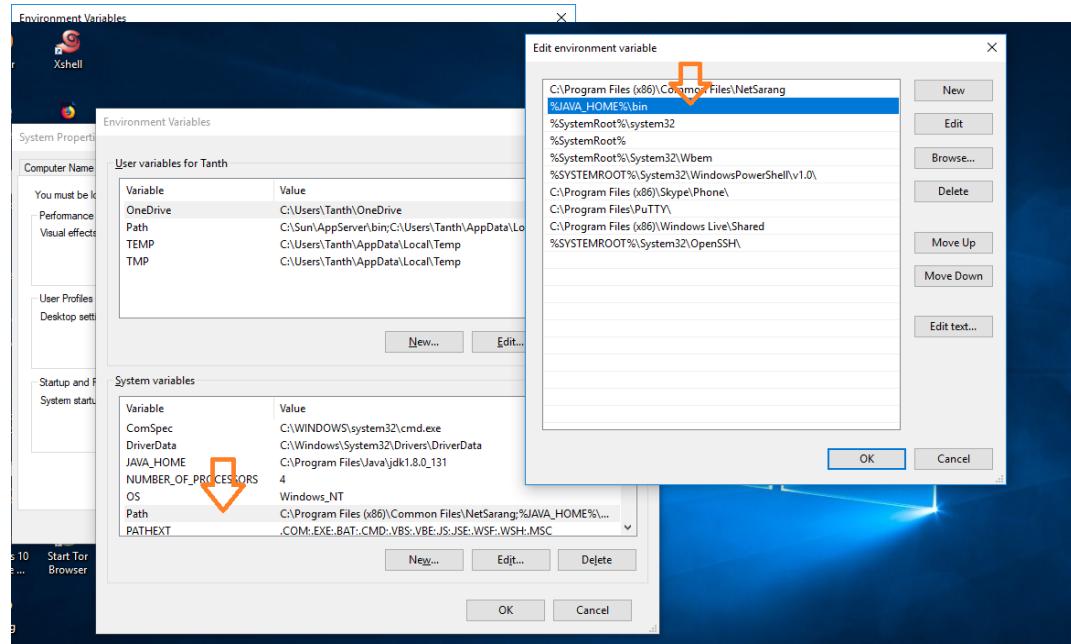
- Mở giao diện như hình bên dưới



- Thêm biến môi trường “JAVA\_HOME”. Giá trị của biến môi trường này là thư mục đã cài JDK có dạng như hình bên dưới.  
“JAVA\_HOME” = “C:\Program Files\Java\jdk1.8.0\_131”.



- Chỉnh sửa biến “Path” : Thêm đường dẫn JDK vào biến Path bằng cách dùng biến môi trường “JAVA\_HOME” như hình bên dưới: "%JAVA\_HOME%\bin"



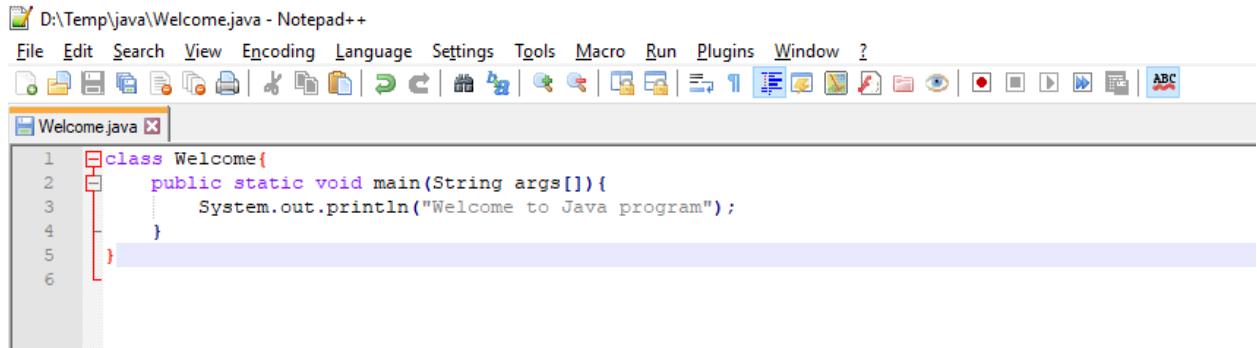
## 2.26.2 Soạn và biên dịch, chạy mã nguồn Java

- Yêu cầu: Tạo chương trình hiển thị dòng chữ “Welcome to Java Basic of Active Study”.

➤ Thực hiện:

- Bước 1: Dùng trình soạn thảo Notepad (Hoặc gì cũng được) để tạo tập tin java nguồn Welcome.java

```
public class Welcome {
    public static void main(String args[]){
        System.out.println("Welcome to Java program");
    }
}
```



- Bước 2: Biên dịch tập tin nguồn.
  - Chọn Start -> Run và nhập lệnh cmd. (Chú ý có thể phải chạy dưới quyền đầy đủ ví dụ như administrator)
  - Trong cửa sổ cmd di chuyển đến thư mục chứa tập tin nguồn cần biên dịch.
  - Trong cửa sổ cmd dùng lệnh javac để biên dịch tập tin nguồn.  
"javac Welcome.java"

Sau khi biên dịch thành công một file có đuôi .class sẽ được sinh ra : "Welcome.class" đây là file dạng Bytecode.

cs: Administrator: Command Prompt

```
D:\Temp\java>dir
Volume in drive D has no label.
Volume Serial Number is 0000-3DB7

Directory of D:\Temp\java <----- Red arrow pointing to the directory output

07/17/2018  06:28 PM    <DIR>      .
07/17/2018  06:28 PM    <DIR>      ..
07/17/2018  06:23 PM            115 Welcome.java <----- Red arrow pointing to the file name
               1 File(s)       115 bytes
               2 Dir(s)  73,460,158,464 bytes free

D:\Temp\java>javac Welcome.java <----- Red arrow pointing to the command

D:\Temp\java>dir
Volume in drive D has no label.
Volume Serial Number is 0000-3DB7

Directory of D:\Temp\java

07/17/2018  06:29 PM    <DIR>      .
07/17/2018  06:29 PM    <DIR>      ..
07/17/2018  06:29 PM            431 Welcome.class <----- Red arrow pointing to the generated class file
               115 Welcome.java
               2 File(s)       546 bytes
               2 Dir(s)  73,460,158,464 bytes free

D:\Temp\java>
```

- Bước 3 Chạy chương trình
  - Cũng trong cửa sổ cmd chạy java với dòng lệnh: “java Welcome”
  - Kết quả thực hiện sẽ hiển thị ngay bên dưới trong cửa sổ cmd

```

Administrator: Command Prompt

D:\Temp\java>javac Welcome.java

D:\Temp\java>dir
Volume in drive D has no label.
Volume Serial Number is 0000-3DB7

Directory of D:\Temp\java

07/17/2018  06:29 PM    <DIR>          .
07/17/2018  06:29 PM    <DIR>          ..
07/17/2018  06:29 PM                431 Welcome.class
07/17/2018  06:23 PM                115 Welcome.java
                           2 File(s)       546 bytes
                           2 Dir(s)   73,460,158,464 bytes free

D:\Temp\java>java Welcome
Welcome to Java program

D:\Temp\java>_

```

### 3 PHẦN 3 JAVA ADVANCE DÀNH CHO LẬP TRÌNH VIÊN

#### 3.1 Kiểu dữ liệu DateTime và formatter

- Để xử lý Date Time trong Java có thể sử dụng một số Class sau:
  - o **java.util.Date** :Lớp được sử dụng rất phổ biến để chứa đựng ngày và giờ, lấy ngày giờ hiện tại, so sánh ngày giờ... Nhiều phương thức của lớp này đã được khuyến cáo không nên sử dụng nữa (**@Deprecated** - bị gạch chéo) và nên sử dụng thay thế bởi các phương thức khác.
  - o **java.util.Calendar** : Lớp abstract cung cấp các phương thức tinh để tính toán với ngày giờ, chẳng hạn như thêm bớt giờ, ngày tháng năm... Lớp này cung cấp đối tượng **static Calendar** bằng cách gọi method **Calendar.getInstance()**; Lưu ý khi dùng đối tượng nay do là đối tượng static vì vậy mọi thay đổi tính toán trên đối tượng đều ảnh hưởng tới các đối tượng còn lại do chúng đều chỉ là một thể hiện (instance) của **Calendar** mà thôi.
  - o **java.util.GregorianCalendar** : Đây là lớp con kế thừa trực tiếp của **Calendar** vì vậy nó cung cấp các khả năng tính toán và có thể khai báo sử dụng như một đối tượng độc lập.

- o **java.util.TimeZone** : Là class cung cấp xử lý với Timezone. Một ngày trên thế giới được chia ra làm 24 TimeZone tương đương với 24 múi giờ. Khi tính toán với thời gian liên quan tới nhiều múi giờ ta sử dụng TimeZone để chỉ ra múi giờ để kết quả tính toán tương ứng với múi giờ địa phương.
- o **java.text.SimpleDateFormat** : Lớp này giúp bạn chuyển đổi qua lại một String có định dạng ngày tháng sang kiểu Date và ngược lại.

### 3.1.1 Nhận Date và Time hiện tại trong Java

```

16 |         //Khai Bao Doi tượng Date
17 |         Date currentTime = new Date();
18 |         //Nhận giá trị date Time thời điểm hiện tại tính theo milisecond
19 |         long lTime = currentTime.getTime();
20 |         //Lấy chuỗi giá trị thời gian hiện tại
21 |         String strTime = currentTime.toString();
22 |         System.out.println("Long Time: " + lTime);
23 |         System.out.println("String Time: " + strTime);

```

run:  
Long Time: 1536567497825  
String Time: Mon Sep 10 15:18:17 ICT 2018  
BUILD SUCCESSFUL (total time: 0 seconds)

- Tại dòng 19 lTime là giá trị thời gian tính theo một giá trị số kiểu long tính từ thời điểm năm 0h ngày 1/1/1970 tới thời điểm giờ hiện tại (GMT +0), quy đổi ra giờ Việt Nam là GMT +7 thi ta cần cộng vào 7h đồng hồ nữa
- Tại dòng 21 strTime là giá trị thời gian được chuyển sang String, giá trị thời gian này đã được tự động quy đổi ra múi giờ hiện tại, nếu ở Việt Nam sẽ là GMT+7.

### 3.1.2 Định dạng Date Time với SimpleDateFormat

- Để chuyển đổi qua lại giữa String và Date ta dùng đối tượng SimpleDateFormat. Đây là đối tượng trung gian quy định định dạng của Date. Vì vậy trước khi thực hiện format date hoặc parse date thì chúng ta cần phải chỉ ra định dạng của Date. Cái này có cú pháp cần thận chúng ta sẽ đi nghiên cứu nó ngay bây giờ
- Để khởi tạo đối tượng SimpleDateFormat chúng ta thực hiện như sau:

```

34  String pattern = "ddMMyyyy HH:mm:ss SSSS";
35  DateFormatSymbols dfs = new DateFormatSymbols(Locale.FRANCE);
36  SimpleDateFormat sdf1 = new SimpleDateFormat();
37  SimpleDateFormat sdf2 = new SimpleDateFormat(pattern);
38  SimpleDateFormat sdf3 = new SimpleDateFormat(pattern,dfs);
39  SimpleDateFormat sdf4 = new SimpleDateFormat(pattern,Locale.FRANCE);
40 ...

```

- Trong đó pattern là định dạng được quy định theo bảng sau:

Letter	Date or Time Component	Presentation	Examples
y	Year	Year	2018; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12

m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

- Một vài ví dụ về Pattern

Pattern	Example
dd-MM-yy	31-01-18
dd-MM-yyyy	31-01-2018
MM-dd-yyyy	01-31-2018
yyyy-MM-dd	2018-01-31
yyyy-MM-dd HH:mm:ss	2018-01-31 23:59:59
yyyy-MM-dd HH:mm:ss.SSS	2018-01-31 23:59:59.999
yyyy-MM-dd HH:mm:ss.SSSZ	2018-01-31 23:59:59.999+0100

- Chuyển đổi từ Date sang String ta dùng method format.

```

42 |     SimpleDateFormat sdf2 = new SimpleDateFormat(pattern);
43 |     String strCurrentDate = sdf2.format(new Date());
44 |     System.out.println("CurrentDate: " + strCurrentDate);

```

Kết quả là:

```
W CurretDate: 13092018 02:40:47 090
```

- Chuyển đổi từ String sang Date ta dùng method parse.

```
46 |     Date currentDate = sdf2.parse("13092018 02:40:05 001");
47 |     System.out.println("Converted Date: " + currentDate.toString());
```

Kết quả là:

```
Converted Date: Thu Sep 13 02:40:05 ICT 2018
```

### 3.2 Kiểu dữ liệu liệt kê Enum Types

- Enum là kiểu dữ liệu giúp lập trình viên tự định nghĩa các giá trị bằng cách liệt kê các ký tự có ý nghĩa gợi nhớ dễ hiểu thay vì các con số khô khan khó hiểu. Cách định nghĩa một kiểu dữ liệu enum ta dùng từ khóa enum:

```
9 |     * @author tanth
10|     */
11|     public enum WeekDay {
12|         MONDAY,
13|         TUESDAY,
14|         WEDNESDAY,
15|         THURSDAY,
16|         FRIDAY,
17|         SATURDAY,
18|         SUNDAY;
19|     }
```

Ví dụ trên ta định nghĩa ra kiểu dữ liệu enum có tên là WeekDay. Kiểu dữ liệu này mang các giá trị ngày từ thứ 2 đến chủ nhật. Các giá trị cách nhau bởi dấu phẩy và kết thúc bằng dấu chấm phẩy.

- Có thể sử dụng toán tử == để so sánh các phần tử enum

```
16 |             WeekDay dayOfWeek = WeekDay.FRIDAY;
17 |             if(dayOfWeek == WeekDay.FRIDAY) {
18 |                 System.out.println("Today is Friday");
19 |             }
```

```
Today is Friday
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Có thể sử dụng với tư cách là tham số với cấu trúc điều khiển switch case

```
21  WeekDay dayOfWeek = WeekDay.FRIDAY;
22  switch (dayOfWeek) {
23      case MONDAY:
24          System.out.println("Today is Monday"); break;
25      case TUESDAY:
26          System.out.println("Today is tuesday"); break;
27      case WEDNESDAY:
28          System.out.println("Today is wednesday"); break;
29      case THURSDAY:
30          System.out.println("Today is thursday"); break;
31      case FRIDAY:
32          System.out.println("Today is friday"); break;
33      case SATURDAY:
34          System.out.println("Today is saturday"); break;
35      case SUNDAY:
36          System.out.println("Today is sunday"); break;
37      default:
38          System.out.println("Today is unknown"); break;
39 }
```

Khi sử dụng các giá trị trong switch case chúng ta không cần phải chỉ ra tên của enum mà Java sẽ tự nhận biết các giá trị tương ứng với biến enum chúng ta sử dụng

- Duyệt các phần tử của enum như là một tập hợp

```
40  for(WeekDay wof: WeekDay.values()) {
41      System.out.println("Today is " + wof);
42 }
```

```
Today is MONDAY
Today is TUESDAY
Today is WEDNESDAY
Today is THURSDAY
Today is FRIDAY
Today is SATURDAY
Today is SUNDAY
```

- Enum cũng có các thuộc tính, method và method khởi động như class nhưng method khởi động chỉ được sử dụng nội bộ trong enum.

```

11  * @author tanth
12  */
13  public enum WeekDay {
14      MONDAY(2),
15      TUESDAY(3),
16      WEDNESDAY(4),
17      THURSDAY(5),
18      FRIDAY(6),
19      SATURDAY(7),
20      SUNDAY(8);
21      int value;
22      WeekDay(int value){
23          this.value = value;
24      }
25
26      public int getValue() {
27          return value;
28      }
29
30      public void setValue(int value) {
31          this.value = value;
32      }
33  }

```

Và enum này vẫn được sử dụng như bình thường giống như các kiểu enum không có constructor khác. Nhưng ta có thể gọi các method khác của enum

```

40  for(WeekDay wof: WeekDay.values()){
41      System.out.println("Today is " + wof.getValue());
42  }

```

Kết quả như sau:

```

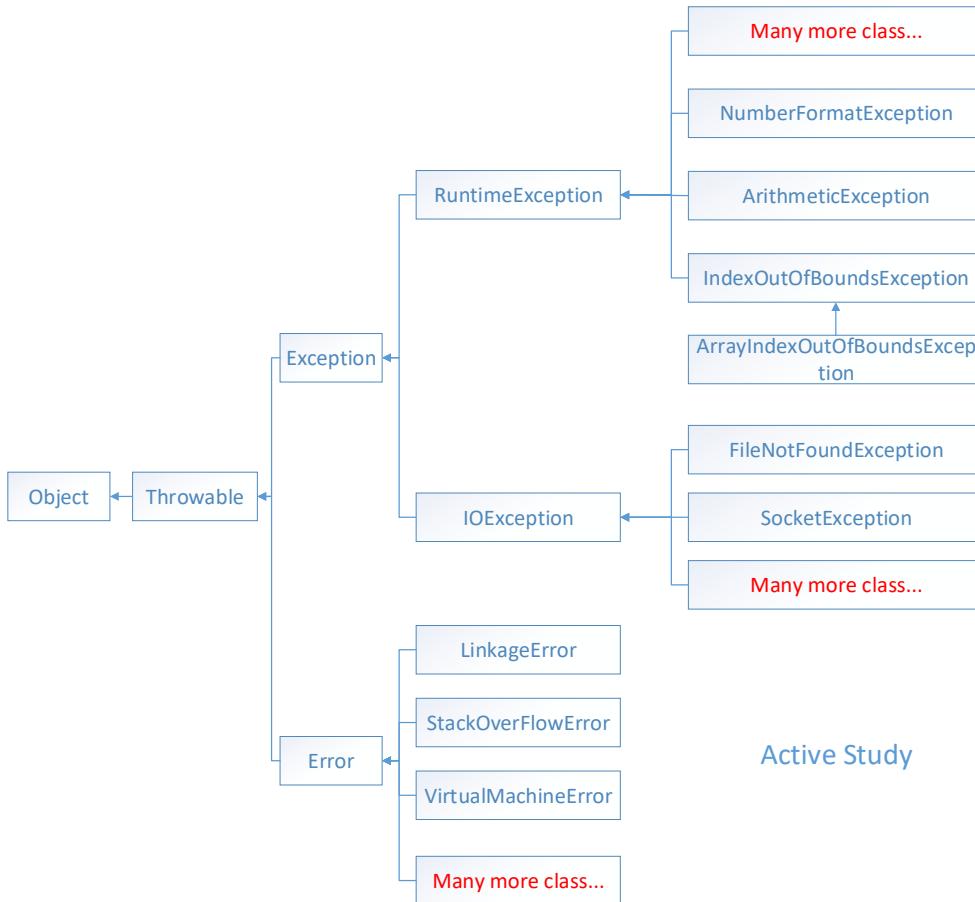
Today is 2
Today is 3
Today is 4
Today is 5
Today is 6
Today is 7
Today is 8
BUILD SUCCESSFUL (total time: 0 seconds)

```

### 3.3 Xử lý ngoại lệ (Exception Handling)

- Trong quá trình chạy thì không có bất cứ một chương trình phần mềm nào có thể tránh được tất cả các lỗi. Các lỗi này có thể dẫn đến chương trình bị dừng đột ngột hoặc chạy sai logic nghiệp vụ...
- Các lỗi có thể xuất phát từ thao tác của người dùng, tài nguyên của hệ thống hoặc lỗi logic của lập trình viên. Các lỗi này phát sinh trong quá trình chạy gọi là lỗi runtime (Runtime error)

- Các lỗi được thể hiện dưới dạng các đối tượng Exception và Error, Exception được tạo ra trong một method và ném ra ngoài lời gọi hàm (throw). Error được tạo ra bởi JVM. Tất cả các Exception và Error đều kế thừa từ lớp Throwable



- Error** là vấn đề nghiêm trọng thường khi xảy ra liên quan tới môi trường thực thi ứng dụng, hệ thống và Error do JVM tạo ra và được đại diện bởi Error Class. Một khi loại lỗi này xảy ra thì là vấn đề vượt quá tầm kiểm soát của lập trình viên và không được xử lý cũng như sẽ được bỏ qua trong quá trình biên dịch và chúng ta chỉ biết thông báo và kết thúc chương trình một cách tốt đẹp nhất mà thôi. Đó là các lỗi như :
  - VirtualMachineError** : Là loại lỗi mà JVM gặp vấn đề do hết tài nguyên hệ thống cho phép để có thể tiếp tục chạy. Tài nguyên hệ thống có thể như RAM...
  - LinkageError** : Là loại lỗi phát sinh khi các thư viện liên kết ngoài bị lỗi hoặc không tìm thấy.
  - StackOverflowError**: là loại lỗi phát sinh khi bộ nhớ stack của chương trình vượt quá ngưỡng cho phép.

- Còn nhiều loại lỗi khác nữa....
- **Exception** là các lỗi được tạo ra ở trong các method và cần được handling tốt trong quá trình viết code. Có 2 loại Exception đó là:
  - **Checked Exception**: có thể được lường trước bởi lập trình viên và trình biên dịch cũng sẽ kiểm tra và yêu cầu cần phải bắt các ngoại lệ này. Đó là các ngoại lệ về **IOException**
  - **Unchecked Exception**: Không thể đoán trước được, đây có thể là các lỗi về logic, trình biên dịch cũng không kiểm tra và đoán trước được vì vậy sẽ bị bỏ qua. Đây là các lỗi thuộc dòng **RuntimeException**.
- Xử lý ngoại lệ trong Java là một kỹ thuật xử lý các lỗi phát sinh trong quá trình chạy chương trình (Runtime Error). Việc này nhằm đảm bảo nếu có xảy ra lỗi ngoại lệ thì luồng xử lý nghiệp vụ vẫn xảy ra theo đúng trình tự logic và mong muốn của lập trình viên.
- Quá trình xử lý các lỗi ngoại lệ được gọi là bắt lỗi (catch Exception). Nếu một lỗi ngoại lệ không được xử lý thì chương trình sẽ kết thúc. Và sau đây là một ví dụ về quá trình bắt lỗi hay còn gọi là handling exception.

```

18   try {
19     System.out.println("Nhập số lượng student: ");
20     Scanner scan = new Scanner(System.in);
21     String strStudent = scan.nextLine();
22
23     int nStudent = Integer.valueOf(strStudent);
24     System.out.println("Số lượng student vừa nhập là: " + nStudent);
25   } catch (Exception ex) {
26     System.out.println("Có lỗi xảy ra: " + ex.getMessage());
27     System.out.println(ex.getClass());
28   }
29   System.out.println("Xin cảm ơn");

```

- Việc bắt lỗi diễn ra trong phạm vi một đoạn hoặc toàn bộ code của method. Được đánh dấu bắt đầu “Try” tới “catch” như ví dụ là bắt đầu từ dòng 18 tới dòng số 24.
- Trong ví dụ trên nếu ta nhập vào số 5 và không có lỗi gì xảy ra thì kết quả sẽ là như sau:

```

run:
Nhập số lượng student:
5
Số lượng student vừa nhập là: 5
Xin cảm ơn

```

- Cũng trong ví dụ trên trong khi chạy người dùng có tình nhập vào ký tự “K” thay vì nhập một chữ số thì kết quả khi thực hiện đoạn mã code trên như sau:

```

run:
Nhap so luong student:
k
Co loi xay ra: For input string: "k"
class java.lang.NumberFormatException
Xin tam biet
BUILD SUCCESSFUL (total time: 12 seconds)

```

- Trong quá trình chạy nếu tại bất kì dòng code nào có thể phát sinh exception thì chương trình sẽ dừng ở dòng đó và bỏ qua tất cả các dòng code tiếp theo và nhảy vào catch gần nhất mà có thể bắt được nó. Nếu không có bất kì catch nào bắt được exception thì sẽ bị throw ra khỏi hàm main và chương trình sẽ dừng lại.
- Khi nhảy vào catch thì chương trình sẽ thực thi tiếp các đoạn mã ở trong block code của catch. Sau khi thực hiện xong tại dòng số 29

Loại lỗi phát sinh đó là NumberFormatException tại dòng số 23 khi cố tình chuyển đổi từ một ký tự sang số

- Để tối ưu quá trình bắt lỗi ta có thể bắt cụ thể NumberFormatException mà ta có thể dự đoán trước thay vì bắt chung dung exception như sau:

```

18 | try {
19 |     System.out.println("Nhap so luong student: ");
20 |     Scanner scan = new Scanner(System.in);
21 |     String strStudent = scan.nextLine();
22 |
23 |     int nStudent = Integer.valueOf(strStudent);
24 |     System.out.println("So luong student vua nhap la: " + nStudent);
25 | }catch (NumberFormatException ex) {
26 |     System.out.println("Co loi xay ra: " + ex.getMessage());
27 |     System.out.println(ex.getClass());
28 | }
29 | System.out.println("Xin tam biet");

```

- Hoặc ta có thể bắt nhiều loại exception khác nhau theo thứ tự từ lớp Exception con tới lớp Exception cha. Ví dụ ta bắt cụ thể NumberFormatException trước ở dòng 25, nếu không phải exception này thì sẽ bắt được exception khác ở dòng số 27 nếu có.

```

18 |     try {
19 |         System.out.println("Nhập số lượng student: ");
20 |         Scanner scan = new Scanner(System.in);
21 |         String strStudent = scan.nextLine();
22 |
23 |         int nStudent = Integer.valueOf(strStudent);
24 |         System.out.println("Số lượng student vừa nhập là: " + nStudent);
25 |     }catch (NumberFormatException ex) {
26 |         System.out.println("Lỗi nhập sai định dạng số : " + ex.getMessage());
27 |     }catch(Exception ex){
28 |         System.out.println("Có lỗi xảy ra: " + ex.getMessage());
29 |         System.out.println(ex.getClass());
30 |     }
31 |     System.out.println("Xin cảm ơn");

```

Và kết quả khi vẫn cố tình nhập sai định dạng chữ thay vì nhập số như sau:

```

run:
Nhập số lượng student:
k
Lỗi nhập sai định dạng số : For input string: "k"
Xin cảm ơn

```

### 3.4 Java IO/NIO.2

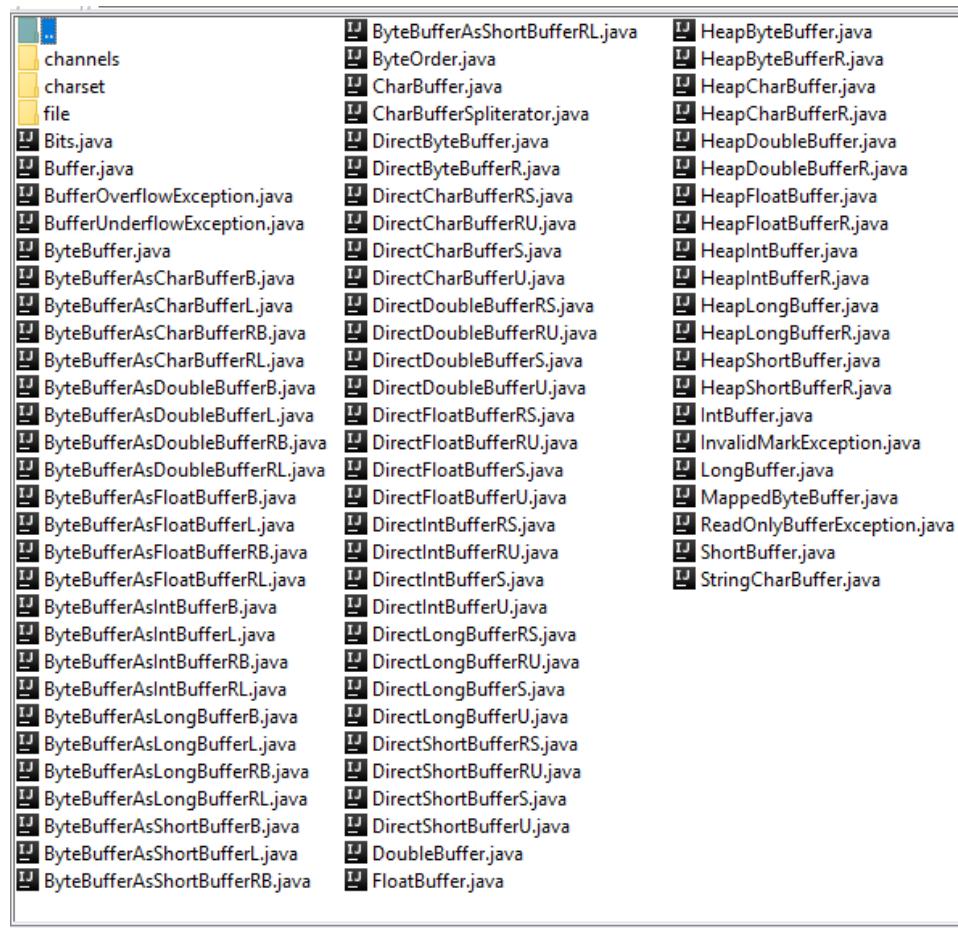
#### 3.4.1 Hiểu biết về Java IO và NIO.2

- Các Class trong java.io:

..		
Bits.java	FileWriter.java	PipedWriter.java
BufferedInputStream.java	FilterInputStream.java	PrintStream.java
BufferedOutputStream.java	FilterOutputStream.java	PrintWriter.java
BufferedReader.java	FilterReader.java	PushbackInputStream.java
BufferedWriter.java	FilterWriter.java	PushbackReader.java
ByteArrayInputStream.java	Flushable.java	RandomAccessFile.java
ByteArrayOutputStream.java	InputStream.java	Reader.java
CharArrayReader.java	InputStreamReader.java	SequenceInputStream.java
CharArrayWriter.java	InterruptedException.java	SerialCallbackContext.java
CharConversionException.java	InvalidClassException.java	Serializable.java
Closeable.java	InvalidObjectException.java	SerializablePermission.java
Console.java	IOException.java	StreamCorruptedException.java
DataInputStream.java	LineNumberInputStream.java	StreamTokenizer.java
DataInputStream.java	LineNumberReader.java	StringBufferInputStream.java
DataOutputStream.java	NotActiveException.java	StringReader.java
DataOutputStream.java	NotSerializableException.java	StringWriter.java
DefaultFileSystem.java	ObjectInputStream.java	SyncFailedException.java
DeleteOnExitHook.java	ObjectInputStreamValidation.java	UncheckedIOException.java
EOFException.java	ObjectOutput.java	UnsupportedEncodingException.java
ExpiringCache.java	ObjectOutputStream.java	UTFDataFormatException.java
Externalizable.java	ObjectOutputStream.java	WinNTFileSystem.java
File.java	ObjectStreamClass.java	WriteAbortedException.java
FileDescriptor.java	ObjectStreamConstants.java	Writer.java
FileFilter.java	ObjectStreamException.java	
FileInputStream.java	ObjectStreamField.java	
FilenameFilter.java	OptionalDataException.java	
FileNotFoundException.java	OutputStream.java	
FileOutputStream.java	OutputStreamWriter.java	
FilePermission.java	PipedInputStream.java	
FileReader.java	PipedOutputStream.java	
FileSystem.java	PipedReader.java	

⇒ Java IO là bộ Package “java.io” trong Java, Nó chứa hầu như tất cả các lớp phục vụ cho việc nhận vào và xuất ra (Input/Output) dữ liệu. Việc nhận vào và xuất ra có thể là các đối tượng File, Screen, socket...

- Các class trong java.nio:



- Trong khi lập trình viên ai cũng biết tới Java IO nhưng ko phải ai cũng biết tới java NIO, bởi chỉ cần java IO cũng đã đủ để lập trình viên làm được gần như mọi thứ chỉ khi nào gấp phải vấn đề với tốc độ và hiệu xuất chương trình thì NIO mới được xem xét và sử dụng tới. NIO là bộ package chứa các lớp phục vụ cho việc làm việc ở tầng sâu hơn với hệ điều hành, chính điều này đã làm cho tốc độ xử lý cao hơn.
- Tuy vậy với phiên bản java 8 trong bộ thư viện IO cũng đã được viết lại và sử dụng những thư viện của NIO, vì vậy tốc độ xử lý của java IO cũng được cải thiện và tối ưu hơn rất nhiều.

### 3.4.2 Làm việc với File trong Java (Handling Files in Java)

- File nói chung có 2 loại là text file và binary file.
  - o Sở dĩ chia ra như vậy là vì định dạng của text file được quy định toàn bộ đều là các mã ký tự được quy định trong bảng mã ASCII hoặc Unicode, và vì vậy nên tất cả những chương trình text editor như **notepad** trong window hay **vi, vim** trong linux đều giúp người dùng đọc và sửa đổi được dạng file này.
  - o Đối với binary file thì dữ liệu được ghi xuống theo đúng bản chất quy định của nó.

- Đối với các ký tự thì không có gì khác biệt giữa 2 loại nhưng đối với kiểu số thì có sự khác biệt vì trong file text tất cả các số đều được quy ra thành các ký tự ví dụ số 123 thì file text sẽ hiểu là 3 ký tự '1', '2' và '3' và quy đổi ra mã Unicode tương ứng là: 49 50 và 51 và ghi 3 giá trị này xuống file text với mỗi giá trị chiếm 2 byte tổng cộng là 6 byte trong file. Cũng với số 123 này ở file nhị phân được hiểu là 1 giá trị số và có giá trị dạng hexa là 7B và được lưu lại trong file giá trị này với 2 byte. Vậy là đối với file nhị phân sẽ tiết kiệm bộ nhớ hơn và đạt hiệu năng cao hơn khi đọc ghi dữ liệu vào ra file.
  - Tất cả file text hay nhị phân khi ghi xuống thiết bị lưu trữ thì bản chất cũng đều là các mã nhị phân 0101 mà thôi, vì vậy việc gọi là file text chỉ mang ý nghĩa quy ước về cách ghi và đọc.
  - File nguồn java cũng là một file dạng text, vì vậy mà các trình soạn thảo code như netbean, Intelij, eclip hay notepad đều có thể đọc được và edit được chúng. Đổi lại file byte code lại là một file dạng nhị phân.
- Chúng ta phân biệt file text và file nhị phân là bởi vì sẽ có những cách đọc và ghi 2 loại file này khác nhau và chúng ta sẽ xem xét ở phần tiếp theo sau đây.
- Class File thuộc thư viện java.io. Class này cho phép chúng ta thao tác với các file hoặc thư mục trong máy tính, ví dụ như tạo file, xóa file, đổi tên hay lấy thông tin file sau đây ta sẽ đi vào ví dụ:
- ```

21 |     File f1 = new File("fileExample.txt");
22 |     try {
23 |         f1.createNewFile();
24 |     } catch (IOException ex) {
25 |         Logger.getLogger(FileExample.class.getName()).log(Level.SEVERE, null, ex);
    |

```
- Sau đây là 3 cách để khởi tạo một đối tượng File:

```

27 |     String pathName = "d:/temp/ActiveStudy";
28 |     String fileName1 = "fileExample1.txt";
29 |     String fileName2 = "fileExample2.txt";
30 |     File f0 = new File(pathName);
31 |     File f1 = new File(pathName, fileName1);
32 |     File f2 = new File(f0, fileName2);
33 |
34 |     try {
35 |         f0.mkdir();
36 |         f1.createNewFile();
37 |         f2.createNewFile();
38 |     } catch (IOException ex) {
39 |         Logger.getLogger(FileExample.class.getName()).log(Level.SEVERE, null, ex);
40 |     }
    |

```

- Tại dòng 30, 31, 32 là 3 cách để tạo ra các đối tượng file.

- Tại dòng 35 thì một thư mục được tạo ra có đường dẫn là "d:/temp/ActiveStudy"
- Tại dòng 36 thì một file có tên "fileExample1.txt" được tạo ra trong thư mục "d:/temp/ActiveStudy"
- Tại dòng 37 thì một file có tên "fileExample2.txt" được tạo ra trong thư mục "d:/temp/ActiveStudy"

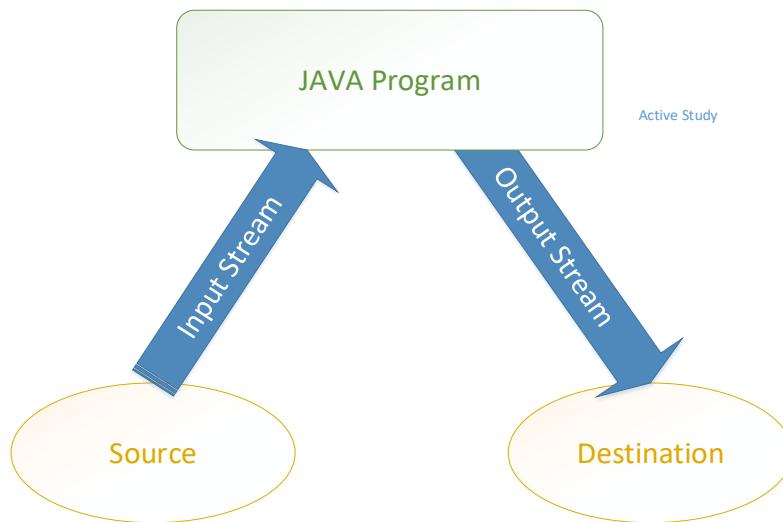
Kết quả như sau:

| This PC > Local Disk (D:) > temp > ActiveStudy |                   |          |      |
|------------------------------------------------|-------------------|----------|------|
| Name                                           | Date modified     | Type     | Size |
| fileExample1.txt                               | 10/4/2018 1:58 AM | TXT File | 0 KB |
| fileExample2.txt                               | 10/4/2018 1:58 AM | TXT File | 0 KB |

- Tuy nhiên ta lại chẳng có thể đọc hay ghi được nội dung file khi chỉ dùng class File. Chúng ta sẽ đi vào nghiên cứu việc đọc và ghi nội dung file ở phần tiếp theo “Stream trong java”

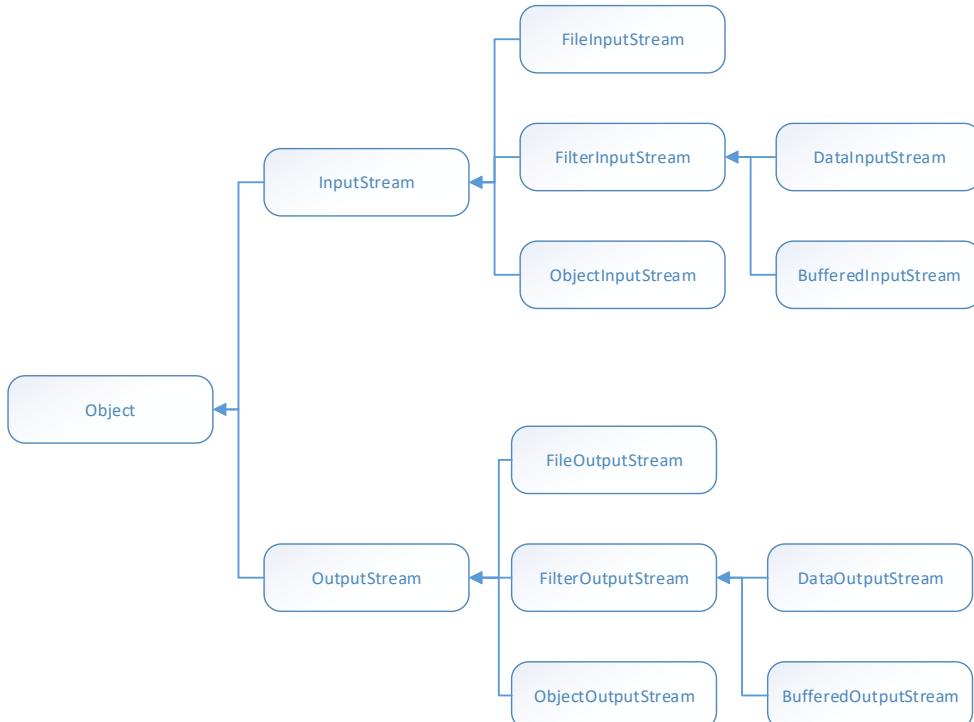
### 3.4.3 Làm việc với Stream trong Java

- Xét hình minh minh họa sau đây mô tả về nhiệm vụ của stream.



- Luồng data được đi từ Source tới một chương trình Java (Java Program). Data có thể lại được đi từ chương trình java tới destination. Trong đó Source và destination có thể là các thiết bị IO như file, Socket, keyboard...
- Vậy hiểu đơn giản Stream là đối tượng giúp data được đọc và ghi ra các thiết bị IO.
- Để đọc hoặc ghi dữ liệu ra các thiết bị thì ta cần thực hiện theo 3 bước sau đây:
  - o 1. Mở một stream

- 2. Đọc Stream cho tới khi đọc hết dữ liệu hoặc ghi dữ liệu vào Stream cho tới khi hết dữ liệu
- 3. Đóng Stream lại.
- Có 2 loại Stream đó là Low level Stream giúp đọc các dữ liệu dạng bytes hoặc ký tự. High level Stream giúp đọc các dữ liệu từ Low level Stream và định dạng dữ liệu trước khi được trả ra.



Trong đó

- `InputStream` và `OutputStream` là các abstract base class cho các class Stream kế thừa và cả hai đều kế thừa từ `Object`
- `FileInputStream`, `FilterInputStream`, `ObjectInputStream` là các Low level Input Stream
- `FileOutputStream`, `FilterOutputStream`, `ObjectOutputStream` là các low level Output Stream
- `DataInputStream`, `BufferedInputStream` là các high level Input Stream
- `DataOutputStream`, `BufferedOutputStream` là các high level Output Stream

Bây giờ ta đã hiểu về Stream, chúng ta sẽ đi tìm hiểu về 2 loại Stream ở phần tiếp theo sau đây

#### 3.4.3.1 Low-Level Streams

Là các Stream để đọc và ghi các dữ liệu dạng byte. Chúng ta xét tới 2 Stream điển hình cho việc đọc ghi dữ liệu low level Stream đó là `FileInputStream`, `FileOutputStream`.

- Chúng ta xét ví dụ về FileOutputStream

```

43     String pathName = "d:/temp/ActiveStudy";
44     String fileName1 = "fileExample1.txt";
45     File f1 = new File(pathName, fileName1);
46     try {
47         FileOutputStream fOutStream = new FileOutputStream(f1);
48         byte[] dataByte = "Active Study".getBytes();
49         fOutStream.write(dataByte);
50     } catch (IOException ex) {
51         Logger.getLogger(FileExample.class.getName()).log(Level.SEVERE, null, ex);
52     }

```

- o Tại dòng 47 một FileOutputStream có tên fOutStream được mở ra
- o Tại dòng 48 chuỗi ký tự “Active Study” được chuyển sang một mảng byte để ghi vào Stream.
- o Tại dòng 49 thực hiện ghi nội dung “Active Study” vào file f1 có tên fileExample1.txt trong thư mục “d:/temp/ActiveStudy”;
- o Tại dòng số 50 chúng ta lưu ý đối với các vào ra IO thì các Exception dạng IO là các checked Exception vì vậy trình biên dịch sẽ check và yêu cầu phải try...catch các exception này.

- Chúng ta xét tiếp ví dụ về FileInputStream

```

54     String pathName = "d:/temp/ActiveStudy";
55     String fileName1 = "fileExample1.txt";
56     File f1 = new File(pathName, fileName1);
57     FileOutputStream fOutStream = null;
58
59     String fileName2 = "fileExample2.txt";
60     File f2 = new File(pathName, fileName2);
61     FileInputStream fInStream = null;
62     try {
63         fOutStream = new FileOutputStream(f1);
64         fInStream = new FileInputStream(f2);
65
66         int c;
67         while ((c = fInStream.read()) != -1) {
68             fOutStream.write(c);
69         }
70         fInStream.close();
71         fOutStream.close();
72     } catch (IOException ex) {
73         Logger.getLogger(FileExample.class.getName()).log(Level.SEVERE, null, ex);
74     }

```

- o Dòng 64 mở ra một InputStream và đọc vào từ file f2 có tên “fileExample2.txt”.

- o Dòng 67 thực hiện đọc vào liên tiếp cho tới khi trả về giá trị -1 và ghi các giá trị đọc được vào file f1 có tên “fileExample1.txt”.
- o Chú ý sau khi đọc xong file thì cần phải đóng hai Stream này lại như dòng 70 và 71;

### 3.4.3.2 High-Level Streams

- Xét ví dụ sau:

- o Giả định danh sách sinh viên có 3 trường dữ liệu gồm sequence, fullName và yearOfBirth được khai báo và khởi tạo tại các dòng 25,26,27.

```

25 |     int[] sequence = {1, 2, 3, 4, 5};
26 |     String[] fullName = {"Nam", "Mo", "Quang", "Manh", "Dung"};
27 |     int[] yearOfBirth = {1997, 1998, 1999, 2000, 2001};

```

- o Tại các dòng 29 là tên file sẽ được ghi dữ liệu sinh viên xuống
- o Tại dòng 30 là khai báo file
- o Tại dòng 31,32 là khai báo một low level input và output Stream

```

29 |     String fileName1 = "d:/temp/ActiveStudy/fileExample1.txt";
30 |     File f1 = new File(fileName1);
31 |     FileOutputStream outPut;
32 |     FileInputStream input;
33 |     DataInputStream dInput;

```

- o Tại dòng 37 khai báo và khởi tạo một high level Output Stream dOutput và có tham số đầu vào của hàm khởi tạo là một low level output Stream đó là outPut;
- o Dòng số 39 chúng ta ghi xuống stream một giá trị sequence có kiểu int
- o Dòng số 40 chúng ta ghi xuống stream một ký tự tab
- o Dòng số 41 chúng ta ghi xuống stream một giá trị fullName có kiểu String Unicode.
- o Dòng số 42 chúng ta lại ghi xuống stream một ký tự tab
- o Dòng số 43 chúng ta ghi xuống stream một giá trị yearOfBirth có kiểu int;
- o Dòng số 44 chúng ta ghi xuống stream một ký tự xuống dòng.
- o Sau cùng tại dòng 46 chúng ta không quên đóng stream lại và dữ liệu sẽ được flush và ghi xuống file(Đoạn này mà quên thì file sẽ ko được ghi dữ liệu xuống.)

```

35     try {
36         outPut = new FileOutputStream(f1);
37         DataOutputStream dOutPut = new DataOutputStream(outPut);
38         for (int i = 0; i < sequence.length; i++) {
39             dOutPut.writeInt(sequence[i]);
40             dOutPut.writeChar('\t');
41             dOutPut.writeUTF(fullName[i]);
42             dOutPut.writeChar('\t');
43             dOutPut.writeInt(yearOfBirth[i]);
44             dOutPut.writeChar('\n');
45         }
46         dOutPut.close();

```

- o Bây giờ chúng ta sẽ đọc dữ liệu lên trả lại vào những biến mảng mới: sequenceCopy, fullNameCopy và yearOfBirthCopy.
- o Tại dòng 49 chúng ta cũng lại khai báo một high level input Stream dInput có tham số truyền vào constructor là một low level input Stream input.
- o Tại dòng 54 kiểm tra nếu dInput Stream còn dữ liệu thì chúng ta lại tiếp tục đọc dữ liệu ra.
- o Dòng số 55 đọc ra một giá trị int đưa vào mảng sequenceCopy;
- o Dòng số 56 đọc vào một ký tự và đây chính là ký tự tab ta đã ghi vào
- o Dòng số 57 đọc vào một chuỗi ký tự Unicode đưa vào mảng fullNameCopy;
- o Dòng số 59 đọc vào một giá trị int đưa vào mảng yearOfBirth;
- o Dòng số 61 in ra màn hình các giá trị đã đọc được.
- o Và cuối cùng không quên đóng Stream tại dòng số 65

```

47 //-----
48     input = new FileInputStream(f1);
49     dInput = new DataInputStream(input);
50     int[] sequenceCopy = new int[5];
51     String[] fullNameCopy = new String[5];
52     int[] yearOfBirthCopy = new int[5];
53     int index = 0;
54     while (dInput.available() > 0) {
55         sequenceCopy[index] = dInput.readInt();
56         dInput.readChar(); //Bỏ qua tab '\t'
57         fullNameCopy[index] = dInput.readUTF();
58         dInput.readChar(); //Bỏ qua tab '\t'
59         yearOfBirthCopy[index] = dInput.readInt();
60         dInput.readChar(); //Bỏ qua ký tự xuống dòng '\n'
61         System.out.println("sequence: " + sequenceCopy[index] + " fullName: "
62                             + fullNameCopy[index] + " yearOfBirth: " + yearOfBirthCopy[index]);
63         index++;
64     }
65     dInput.close();
66 } catch (FileNotFoundException ex) {
67     Logger.getLogger(DataInputStreamExample.class.getName()).log(Level.SEVERE, null, ex);
68 } catch (IOException ex) {
69     Logger.getLogger(DataInputStreamExample.class.getName()).log(Level.SEVERE, null, ex);
70 }

```

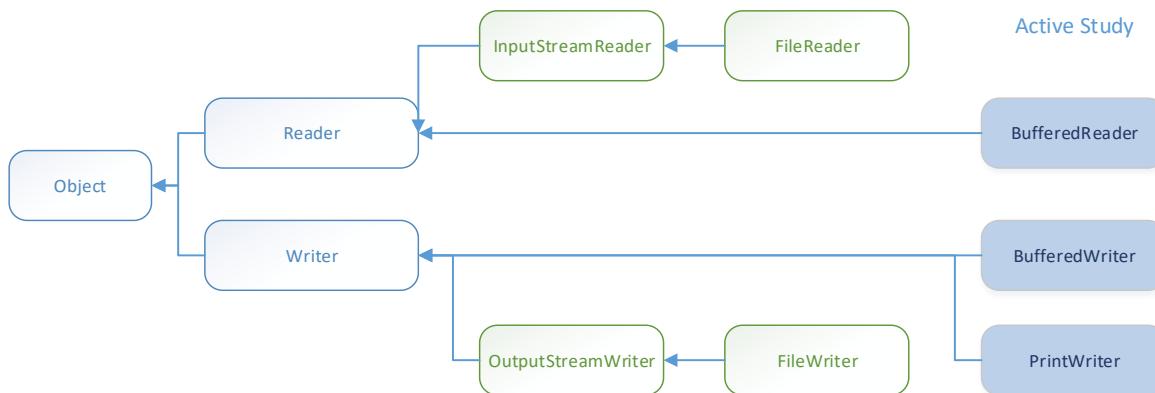
- Kết quả khi chạy chương trình như sau:

```
run:
sequence: 1 fullName: Nam yearOfBirth: 1997
sequence: 2 fullName: Mo yearOfBirth: 1998
sequence: 3 fullName: Quang yearOfBirth: 1999
sequence: 4 fullName: Manh yearOfBirth: 2000
sequence: 5 fullName: Dung yearOfBirth: 2001
-----
```

- Như vậy có thể dễ dàng nhận thấy Việc đọc dữ liệu ra theo đúng như trình tự và định dạng dữ liệu lúc chúng ta ghi vào. Khi ghi vào tùy theo quy ước của chúng ta và lúc đọc ra cũng phải theo quy ước này.
- Các high level Stream giúp chúng ta ghi vào và đọc ra chính xác theo từng loại dữ liệu.
- Các high level Stream cũng không tự mình mở ra một IO device mà phải đọc và ghi dữ liệu thông qua một low level Stream (dòng số 37 và 49 cho thấy điều này).

#### 3.4.4 Readers and Writers

- Reader và Writer sẽ giúp chúng ta đơn giản hơn trong việc đọc và nội dung một file text và chúng được kế thừa từ các class reader và writer chứ không phải từ InputStream và OutputStream.



##### 3.4.4.1 Low-Level Readers and Writers

- Các low level reader đó là FileWriter và FileReader giúp đọc và ghi các ký tự vào file.
- Xét ví dụ dưới đây về cách đọc và ghi file dùng hai đối tượng này.

```

22  File f1 = new File("d:/temp/ActiveStudy/fileExample1.txt");
23  String data = "Xin chào mừng các bạn đã đến với Active Study";
24  try {
25      FileWriter fwrite = new FileWriter(f1);
26      fwrite.write(data);
27      fwrite.close();
28      FileReader fread = new FileReader(f1);
29      int singleChar = 0;
30      while(singleChar != -1){
31          singleChar = fread.read();
32          System.out.print((char)singleChar);
33      }
34      fwrite.close();
35  } catch (FileNotFoundException ex) {
36      Logger.getLogger(ReaderExample.class.getName()).log(Level.SEVERE, null, ex);
37  } catch (IOException ex) {
38      Logger.getLogger(ReaderExample.class.getName()).log(Level.SEVERE, null, ex);
39  }

```

### 3.4.4.2 High-Level Readers and Writers

- Các high level reader và writer đó là BufferedReader và BufferedWriter được ưu tiên sử dụng trong trường hợp chúng ta cần đọc và ghi nhanh một lượng lớn các dòng text xuống file.
- Ví dụ dưới đây là cách dùng với BufferedReader và BufferedWriter.
  - o Tại dòng 49 đối tượng bReader được khai báo và khởi tạo và được truyền vào một low level reader.
  - o Để ý kỹ ta thấy bReader không được đóng sau khi dùng xong, điều này thoạt nhìn có vẻ gây ra vấn đề thất thoát tài nguyên vì bất kể một resource gì khi mở thì đều phải đóng. Tuy nhiên bReader vẫn được đóng bởi vì nó đã tự được đóng khi dùng xong hoặc khi xảy ra một exception bởi Try catch, bởi vì bReader là tài nguyên được khai báo trong try() như dòng 49. Nhớ là trong Try(...) chứ không phải trong Body của Try{}catch. Vì vậy try...catch sẽ quản lý để đóng tài nguyên này khi chúng ta dùng xong. Cách này nên được ưu tiên sử dụng vì tính an toàn của chúng. Tài nguyên sẽ được đóng và giải phóng trong bất kì trường hợp nào kể cả bị exception.

```

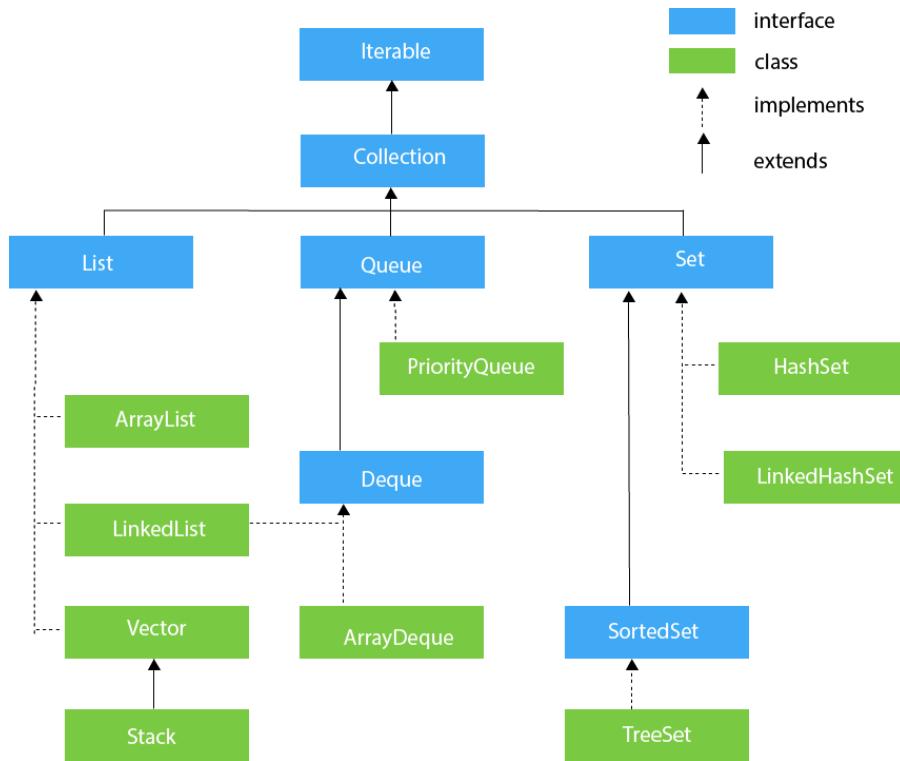
45     File f1 = new File("d:/temp/ActiveStudy/fileExample1.txt");
46     File f2 = new File("d:/temp/ActiveStudy/fileExample2.txt");
47     BufferedWriter bWriter;
48     try {
49         try (BufferedReader bReader = new BufferedReader(new FileReader(f1))) {
50             bWriter = new BufferedWriter(new FileWriter(f2));
51             String line;
52             while ((line = bReader.readLine()) != null) {
53                 bWriter.write(line);
54                 bWriter.newLine();
55             }
56         }
57         bWriter.close();
58     } catch (FileNotFoundException ex) {
59         Logger.getLogger(ReaderExample.class.getName()).log(Level.SEVERE, null, ex);
60     } catch (IOException ex) {
61         Logger.getLogger(ReaderExample.class.getName()).log(Level.SEVERE, null, ex);
62     }

```

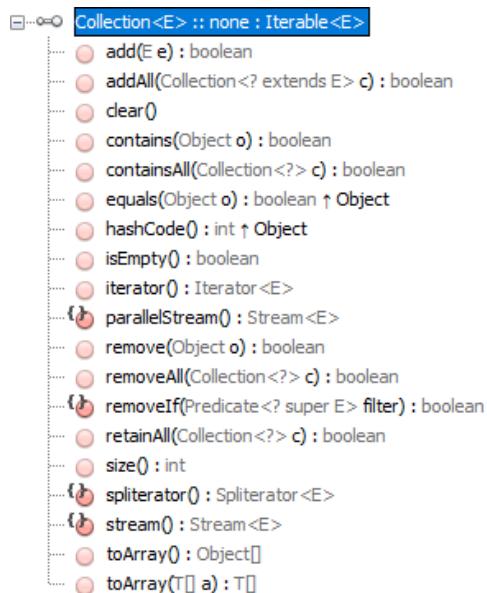
### 3.5 Collection

#### 3.5.1 Collection Interface trong Java

- Chúng ta đã làm quen với kiểu dữ liệu mảng(Array). Tuy đây là một hình thức tập hợp của nhiều dữ liệu nhưng lại không phải là một kiểu dữ liệu Collection bởi lẽ nó không kế thừa từ một Collection.
- Lấy cảm hứng từ kiểu dữ liệu Array thì Collection ra đời để lưu trữ và thao tác với các dữ liệu này được nhanh chóng, thuận tiện và an toàn hơn.
- Căn cứ vào mô hình phân cấp dưới đây ta nhận thấy **Collection Interface** là một Interface gốc (Root Interface)
- Cũng theo phả hệ kế thừa thì các Interface List, Queue, Set kế thừa trực tiếp từ Collection Interface.



- Collection Interface quy định các phương thức cơ sở để xử lý với một kiểu dữ liệu tập hợp ví dụ như thêm vào, xóa đi, tìm kiếm, kiểm tra kích cỡ tập hợp phần tử dữ liệu... như hình ở bên dưới.



| Method                         | Description                                                                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean add(E e);</code> | Thêm một phần tử vào Collection và nếu thành công sẽ trả về ngay lập tức giá trị <b>True</b> . Nếu không thành công sẽ trả về <b>False</b> . |

|                                                              |                                                                                                                                                       |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean addAll(Collection&lt;? extends E&gt; c)</code> | Thêm vào nhiều phần tử từ một collection c khác. Trả về giá trị True nếu Collection thay đổi sau lời gọi method                                       |
| <code>void clear()</code>                                    | Xóa tất cả các phần tử của collection.                                                                                                                |
| <code>boolean contains(Object o)</code>                      | Trả về giá trị true nếu collection chứa phần tử có giá trị bằng o. Ngược lại trả ra giá trị false.                                                    |
| <code>boolean containsAll(Collection&lt;?&gt; c);</code>     | Trả về giá trị true nếu collection chứa toàn bộ các phần tử của collection c. Ngược lại trả ra giá trị false                                          |
| <code>boolean isEmpty()</code>                               | Trả ra giá trị true nếu collection không có phần tử nào, ngược lại trả ra giá trị false.                                                              |
| <code>boolean removeAll(Collection&lt;?&gt; c);</code>       | Xóa khỏi Collection tất cả các phần tử có giá trị bằng các phần tử trong collection c. Trả về giá trị True nếu Collection thay đổi sau lời gọi method |
| <code>boolean retainAll(Collection&lt;?&gt; c)</code>        | Xóa bỏ toàn bộ các phần tử của collection không tồn tại trong collection c. Trả về giá trị true nếu collection có sự thay đổi sau lời gọi hàm.        |
| <code>int size();</code>                                     | Trả về số lượng phần tử trong collection.                                                                                                             |
| <code>Object[] toArray()</code>                              | Trả về mảng các Object của các phần tử trong collection.                                                                                              |
| <code>&lt;T&gt; T[] toArray(T[] a)</code>                    | Trả về mảng các đối tượng có kiểu T của các phần tử trong collection. T là kiểu dữ liệu được chỉ ra khi khởi tạo collection.                          |

- Do bản thân Collection là một interface vì vậy ta không thể khởi tạo một đối tượng Collection nhưng ta có thể khai báo một biến có kiểu Collection như bên dưới như sau:

```

26 |   Collection numbers;
27 |   Collection<Integer> numberInts;
28 |   Collection<Student> students;
29 |

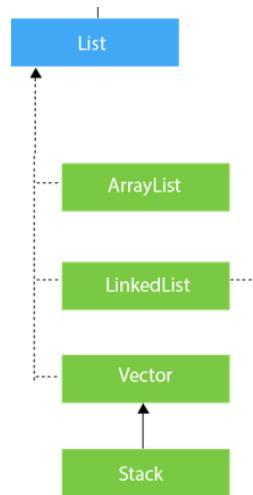
```

- o Trong đó <Student> là kiểu dữ liệu, Nó chỉ định Collection này sẽ chứa đựng các phần tử có kiểu là Student.
  - Nếu đã chỉ ra một kiểu dữ liệu trong dấu ngoặc <> thì trong quá trình sử dụng ta thêm các phần tử dạng khác (không thuộc phả hệ con của Student) thì lập tức sẽ bị báo lỗi. Hoặc khi truy xuất tới các phần tử thì ta sẽ được trả ra đúng kiểu dữ liệu được quy định trong <> mà không cần phải cast (ép kiểu)
  - Mọi Kiểu dữ liệu đều cho phép trừ 8 kiểu dữ liệu nguyên thủy (primitive)
  - Nếu không chỉ ra kiểu dữ liệu nào và cũng ko có dấu ngoặc <> thì biến Collection sẽ chấp nhận tất cả các kiểu dữ liệu và khi truy xuất tới các phần tử này thì kiểu dữ liệu trả ra cũng là kiểu Object và chúng ta phải cast để được đúng kiểu dữ liệu mong muốn.
  - Phần này chúng ta sẽ có ví dụ cụ thể hơn ở phần ArrayList để hiểu rõ hơn.
- o students là tên của biến có kiểu Collection.

- Kích cỡ các phần tử của Collection sẽ lớn dần lên khi chúng ta add vào các phần tử. vì vậy chúng ta không cần phải khai báo số lượng các phần tử ngay từ đầu. Điều này làm cho chương trình của chúng ta có được chiến lược sử dụng bộ nhớ hiệu quả và giúp chúng ta không phải lo nghĩ hoặc nghỉ ngơi vấn đề thiếu số lượng phần tử khi thêm vào.
- Sau đây chúng ta sẽ đi xét tới các interface quan trọng khác kế thừa từ Collection Interface để thấy chức năng và nhiệm vụ chuyên biệt của từng loại.

### 3.5.2 List Interface và các lớp thực thi của List trong Java

- List là một kiểu dữ liệu Collection quan trọng và sử dụng nhiều vào bậc nhất trong lập trình Java.
- Các class ArrayList, LinkedList, Vector là các lớp con trực tiếp implement (kế thừa thực thi) interface này.



- List: là tập hợp các phần tử dữ liệu có thể được truy cập thông qua các chỉ mục (ordered) và cho phép trùng lặp (Duplicate Element)

#### 3.5.2.1 ArrayList

- Xét ví dụ sau:

```

33   ArrayList<Float> arrNumbers = new ArrayList();
34   arrNumbers.add(5.5f);
35   float x = arrNumbers.get(0);
36   arrNumbers.add(6f);
37   float y = arrNumbers.get(1);
38   System.out.println("x = " + x);
39   System.out.println("y = " + y);
40
  
```

- Tại dòng 33 ta khai báo và khởi tạo một biến đối tượng `arrNumber` kiểu ArrayList và chỉ định kiểu dữ liệu mà arrNumber sẽ chứa đựng là Float
- Tại dòng 34 ta thêm vào một đối tượng Float có giá trị 5.5f. Giá trị này AutoBoxing sang Float. lúc này giá trị sẽ được đưa vào vị trí đầu tiên có index = 0;
- Tại dòng 35 ta lấy ra giá trị tại vị trí thứ 0 (Index = 0) và đưa vào biến x
- Tại dòng 36 ta tiếp tục đưa thêm một giá trị vào, lúc này giá trị sẽ được đưa vào vị trí tiếp theo có index = 1;
- Tại dòng 37 ta lấy ra giá trị tại vị trí thứ 1 (Index = 1) và đưa vào biến y
- Sau đó hiển thị ra các giá trị x và y như sau:

: Output - JavaCoreBasic (run)

```

run:
x = 5.5
y = 6.0

```

- Ta xét thêm về việc remove một phần tử trong ArrayList

```

41      arrNumbers.remove(0);
42      float z = arrNumbers.get(0);
43      System.out.println("z = " + z);
44

```

- Tại dòng 41 thì phần tử có chỉ số 0 được remove ra khỏi arrNumbers
- Tại dòng 42 khi truy cập lại tới phần tử tại vị trí 0 thì vẫn khả thi vì các phần tử phía sau được đánh lại chỉ số vì vậy chúng được dồn lại và lại bắt đầu từ 0.
- Tại dòng 43 kết quả truy cập tới phần tử 0 ta được kết quả z = 6.0.

: Output - JavaCoreBasic (run)

```

run:
x = 5.5
y = 6.0
z = 6.0

```

- Nếu chúng ta tiếp tục truy cập tới phần tử mà không tồn tại chỉ số đó thì sẽ nhận về một Exception vì vậy cần chú ý chỉ số truy cập như đối với Array.

: Output - JavaCoreBasic (run) ×

```

run:
x = 5.5
y = 6.0
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 1, Size: 1

```

- Đối tượng ArrayList cung cấp khả năng lưu trữ tuyến tính các phần tử như một mảng dữ liệu, nó cho phép thêm, xóa bỏ, truy cập tới các phần tử thông qua chỉ số (Index).
- Nếu cần thêm hoặc truy cập thường xuyên tới các phần tử thì ta nên sử dụng kiểu dữ liệu ArrayList.
- Nếu cần insert hoặc remove các phần tử một cách thường xuyên thì ta không nên sử dụng ArrayList bởi vì việc truy cập tới các phần tử ArrayList là tuyến tính vì vậy quá trình Insert và remove sẽ chạy chậm hơn. Trong trường hợp này ta nên sử dụng LinkedList.

### 3.5.2.2 *LinkedList*

- Xét ví dụ sau về việc khai báo và sử dụng LinkedList

```

16
17     LinkedList<Integer> lktNumber = new LinkedList();
18     lktNumber.add(100);
19     lktNumber.add(101);
20     lktNumber.add(102);
21     lktNumber.removeFirst();
22     lktNumber.removeLast();
23     System.out.println("The first element is: " + lktNumber.getFirst());
24

```

- o Dòng 17 khai báo LinkedList có tên lktNumber để chứa đựng các dữ liệu Integer.
- o Dòng 18,19,20 lần lượt thêm các giá trị vào lktNumber
- o Dòng 20 và 21 remove 2 phần tử ở đầu và cuối danh sách.
- o Dòng 23 lấy và đưa ra giá trị của phần tử đầu tiên trong danh sách, và đây là kết quả:

The screenshot shows the Java IDE's output window titled "Output - JavaCoreBasic (run)". It contains two entries: a green play button icon followed by the text "run:" and a yellow play button icon followed by the text "The first element is: 101".

- LinkedList cũng có cách khai báo và sử dụng giống với ArrayList.
- Đối tượng LinkedList cung cấp khả năng lưu trữ các phần tử theo một dạng danh sách liên kết. Mỗi một phần tử (node) trong danh sách chứa đựng một đối tượng data và trỏ tới phần tử (node) tiếp theo. Vì vậy khi cần truy cập ngẫu nhiên tới các phần tử thì LinkedList sẽ phải duyệt các phần tử từ đầu node cho tới vị trí cần truy cập, điều này nói lên rằng việc truy cập ngẫu nhiên sẽ chậm chạp hơn nhiều so với ArrayList.
- Đối với việc remove và insert phần tử trong LinkedList thì lại thuận tiện và nhanh chóng bởi vì chỉ cần điều chỉnh vại những vị trí node muốn insert hoặc remove.

### 3.5.2.3 *Vector*

- Ta xét ví dụ sau về việc khai báo và sử dụng một Vector.

Website: <http://activestudy.online>

Fanpage: <https://www.facebook.com/activestudy.edu.vn>

Support Group (Active Study member only): <https://www.facebook.com/groups/ActiveStudy.JavaCoreBasic.Support>

```

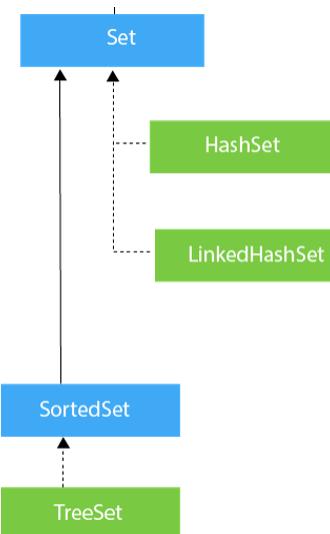
17
18     Vector<Integer> vNumbers = new Vector<>();
19     vNumbers.add(1000);
20

```

- Vector khá giống với ArrayList. Tuy nhiên Vector được thiết kế để chạy trong môi trường Multi Thread do nó được đồng bộ (Synchronization) vì vậy nó an toàn (Thread safe)
- Nếu không sử dụng Multi thread thì không nên sử dụng Vector vì nó sẽ làm chậm và giảm hiệu suất chương trình của chúng ta.

### 3.5.3 Set Interface và các lớp thực thi của Set trong Java

- Là tập hợp các phần tử dữ liệu và không cho phép trùng lặp (not Duplicate Element)



#### 3.5.3.1 HashSet

- Xét ví dụ sau về việc khai báo và sử dụng một HashSet.

```

17
18     HashSet<Integer> hasNumbers = new HashSet<>();
19     hasNumbers.add(1000);
20     hasNumbers.add(1001);
21     hasNumbers.add(1002);
22
23     for(Integer x : hasNumbers){
24         System.out.println("The elements is: " + x);
25     }

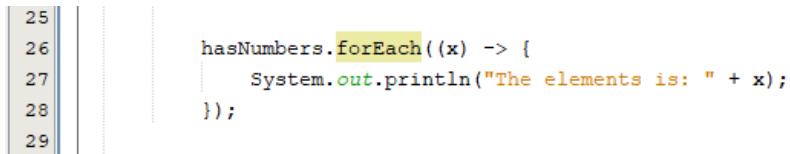
```

- o Dòng 18 khai báo và cấp phát một biến đối tượng hasNumbers
- o Dòng 19 đến 21 chúng ta thêm vào 3 phần tử Integer có giá trị 1000, 1001 và 1002. Method này sẽ trả về giá trị true nếu phần tử chưa tồn tại trong danh sách và trả về false và danh sách không thay đổi gì khi thêm vào một giá trị đã tồn tại trong danh sách.

- Dòng 22 sử dụng for-each để duyệt lần lượt các phần tử của hashNumber và in giá trị của chúng ra ta được kết quả như sau:

```
Output - JavaCoreAdvance (run) ×
run:
The elements is: 1000
The elements is: 1001
The elements is: 1002
```

- Mở rộng: Ngoài cách tại dòng 22 để truy cập tới các phần tử của HashSet ta còn có thể sử dụng biểu thức lambda để thực hiện như sau. Biểu thức này chỉ có ở Java 8 trở đi.

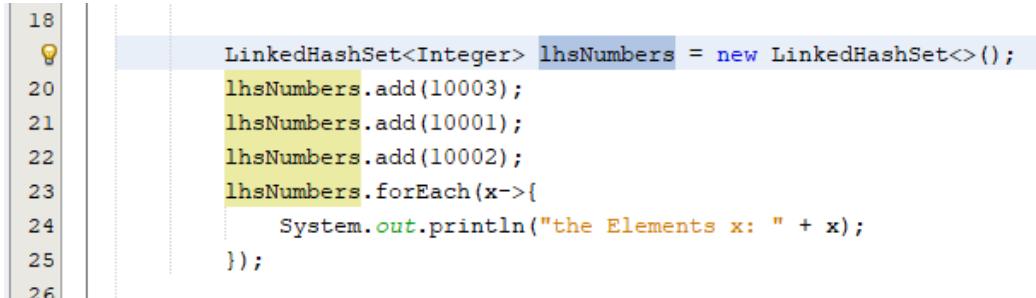


```
25
26     hasNumbers.forEach((x) -> {
27         System.out.println("The elements is: " + x);
28     );
29 }
```

- HashSet là kiểu dữ liệu tập hợp có tốc độ truy cập nhanh, nhanh hơn TreeSet
- HashSet không cho phép các phần tử dữ liệu có giá trị trùng nhau.
- Do HashSet là kiểu dữ liệu không được order(không có thứ tự) vì vậy chúng ta không thể truy cập tới các phần tử thông qua chỉ số của chúng và không đảm bảo theo một trật tự nhất định nào cả.
- HashSet không được thiết kế để chạy an toàn trong môi trường multi thread. Vì vậy để an toàn thì có thể sử dụng phương thức synchronization của class Collections.

### 3.5.3.2 *LinkedHashSet*

- Ta xét ví dụ sau về LinkedHashSet để xem cách khai báo và sử dụng chúng.



```
18
19     LinkedHashSet<Integer> lhsNumbers = new LinkedHashSet<>();
20     lhsNumbers.add(10003);
21     lhsNumbers.add(10001);
22     lhsNumbers.add(10002);
23     lhsNumbers.forEach(x->{
24         System.out.println("the Elements x: " + x);
25     );
26 }
```

- Dòng 19 khai báo và cấp phát đối tượng cho biến đối tượng lhsNumbers
- Dòng 20 đến 22 thêm vào các phần tử. Method này sẽ trả về giá trị true nếu phần tử chưa tồn tại trong danh sách và trả về false và danh sách không thay đổi gì khi thêm vào một giá trị đã tồn tại trong danh sách.
- Dòng 23, 24 để hiển thị từng phần tử trong danh sách.

```
Output - JavaCoreAdvance (run) ×
run:
the Elements x: 10003
the Elements x: 10001
the Elements x: 10002
```

- Giống như HashSet nhưng LinkedHashSet duy trì một liên kết đôi giữa các phần tử dữ liệu. Các phần tử có trật tự (ordered) và trật tự được xác định khi chèn phần tử vào danh sách. Nhưng chúng không được sắp xếp.
- LinkedHashSet không được thiết kế để làm việc an toàn trong môi trường multi Thread. Để an toàn thì có thể sử dụng phương thức synchronization của class Collections.

### 3.5.3.3 TreeSet

- Chúng ta xét ví dụ sau về cách khai báo và sử dụng TreeSet.

```
16
17     TreeSet<Integer> trsNumbers = new TreeSet();
18     trsNumbers.add(1003);
19     trsNumbers.add(1001);
20     trsNumbers.add(1002);
21     Integer x = trsNumbers.first();
22     Integer y = trsNumbers.last();
23     trsNumbers.forEach(z ->{
24         System.out.println("The elements is: " + z);
25     });
26
```

- o Tại dòng 17 chúng ta khai báo một biến đối tượng trsNumbers kiểu TreeSet và cấp phát đối tượng.
- o Tại dòng 18 đến 20 chúng ta add lần lượt các giá trị vào trsNumbers. Method này sẽ trả về giá trị true nếu phần tử chưa tồn tại trong danh sách và trả về false và danh sách không thay đổi gì khi thêm vào một giá trị đã tồn tại trong danh sách.
- o Tại dòng 21 và 22 chúng ta truy cập tới các phần tử đầu và cuối của trsNumbers.
- o Tại dòng 23 chúng ta truy cập tới lần lượt các phần tử của trsNumbers thông qua for-each. Và kết quả như sau. Chúng đã được sắp xếp từ theo chiều tăng dần giá trị mặc dù chúng ta không hề sắp xếp chúng.

```
Output - JavaCoreAdvance (run) ×
run:
The elements is: 1001
The elements is: 1002
The elements is: 1003
```

- o Nếu chúng ta muốn TreeSet được sắp xếp theo thứ tự giảm dần thì ta thực hiện như sau:

```

26
27     trsNumbers.descendingSet();
28     trsNumbers.forEach(z ->{
29         System.out.println("The elements is: " + z);
30     });
31

```

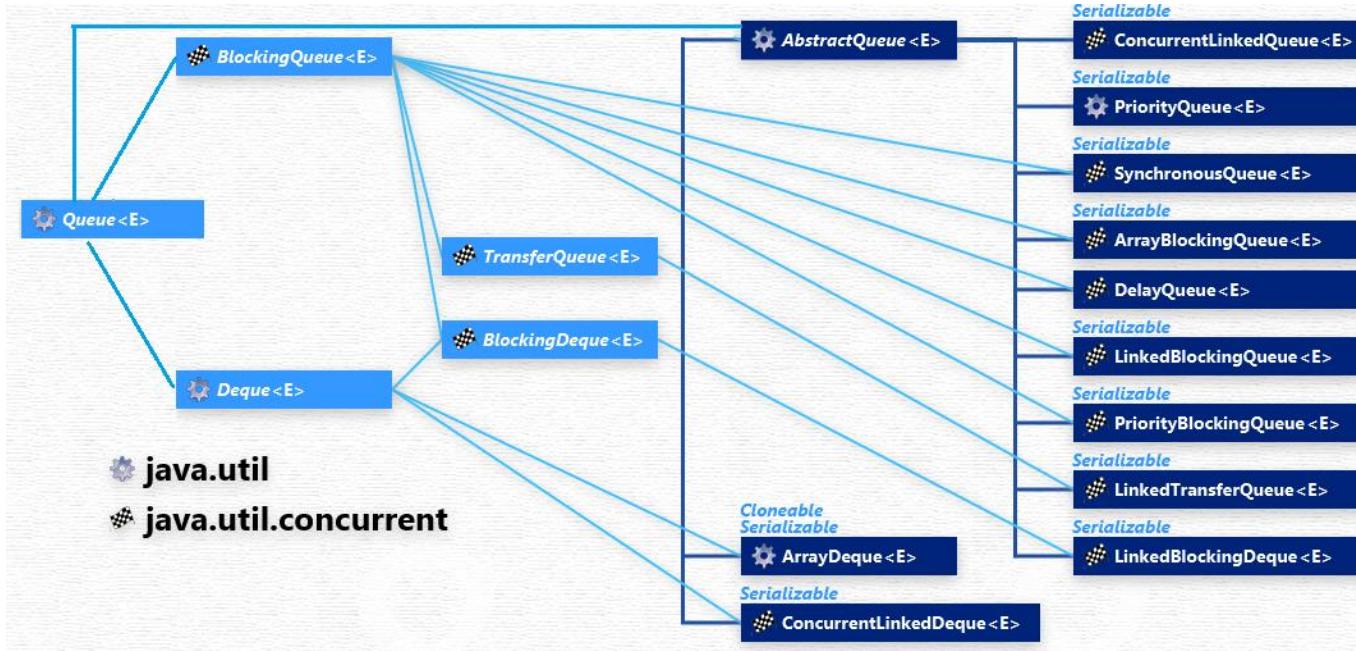
Và kết quả là:

The screenshot shows the Java Core Advance project running. The output window displays the following text:  
run:  
The elements is: 1001  
The elements is: 1002  
The elements is: 1003

- Các phần tử trong TreeSet đều đã được sắp xếp tự động, chính vì điều này làm cho quá trình truy cập tới các thành phần dữ liệu của TreeSet trở nên chậm chạp hơn HashSet.
- TreeSet không cho phép các phần tử dữ liệu có giá trị trùng nhau.
- TreeSet không được thiết kế để chạy an toàn trong môi trường Multi thread. Chính vì vậy để an toàn thì có thể sử dụng phương thức synchronization của class Collections.
- Chúng ta nên sử dụng TreeSet khi cần phải thao tác với một danh sách luôn luôn cần sắp xếp một cách tự động (Tăng hoặc giảm dần là tùy ở thuật toán của chúng ta) nhưng vẫn đề về tốc độ thêm vào danh sách các phần tử lại không phải là cần thiết lắm.

### 3.5.4 Queue Interface trong Java

- **Queue** (hàng đợi) là một Interface kế thừa từ Interface **Collection**, nó có đầy đủ các tính năng của **Collection**, và khá giống với List, tuy nhiên mục đích sử dụng lại khác nhau.
- **Queue** hoạt động theo cách thức **FIFO** (First In First Out). Nghĩa là chúng ta chỉ có thể truy cập phần tử ở đầu hàng đợi, và khi loại bỏ phần tử nó chỉ loại phần tử đứng ở đầu hàng đợi. Nó giống như hàng người xếp hàng ở rạp chiếu film, chỉ người đứng đầu hàng đợi mới được phục vụ bán vé, người mới đến sẽ được chèn vào hàng đợi, vị trí được chèn vào có thể không phải là cuối hàng. Vì trí phần tử được chèn vào phụ thuộc vào loại hàng đợi và độ ưu tiên của phần tử đó.
- Queue cho phép các phần tử trùng lặp và Không cho phép phần tử null.



| Method                           | Description                                                                                                                                                             |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>boolean add(E e);</code>   | Thêm một phần tử vào Queue và nếu thành công sẽ trả về ngay lập tức giá trị <code>True</code> . Nếu không thành công sẽ trả về một <code>IllegalStateException</code> . |
| <code>boolean offer(E e);</code> | Thêm một phần tử vào Queue và trả về ngay lập tức giá trị <code>True</code> hoặc <code>False</code> .                                                                   |
| <code>E remove();</code>         | Trả ra và đồng thời xóa bỏ phần tử đầu ra khỏi Queue.<br>Nếu Queue rỗng thì sẽ trả ra một exception <code>NoSuchElementException</code>                                 |
| <code>E poll();</code>           | Trả ra và đồng thời xóa bỏ phần tử đầu ra khỏi Queue.<br>Nếu Queue rỗng thì sẽ trả ra <code>Null</code>                                                                 |
| <code>E element();</code>        | Trả ra và không xóa bỏ phần tử đầu ra khỏi Queue.<br>Nếu Queue rỗng thì sẽ trả ra một exception <code>NoSuchElementException</code>                                     |
| <code>E peek();</code>           | Trả ra và không xóa bỏ phần tử đầu ra khỏi Queue.<br>Nếu Queue rỗng thì sẽ trả ra <code>Null</code>                                                                     |

- Queue được dùng để chứa dữ liệu và xử lý dữ liệu theo thuật toán FIFO (First In First Out).
- Một ứng dụng quan trọng nữa của Queue là để đồng bộ tốc độ xử lý dữ liệu giữa các tiểu tiến trình (Thread) hoặc giữa các module. Bởi vì các thread có tốc độ xử lý khác nhau vì vậy chúng cần phải được trao đổi dữ liệu với nhau thông qua một Queue. Lúc này Queue có thể được hiểu là một bộ đệm dữ liệu (buffer) để các thread và module không phải đợi chờ nhau trong việc xử lý dữ liệu của nhau.
- Ứng dụng quan trọng khác của Queue là được dùng để làm nơi trao đổi dữ liệu chung của nhiều thread, ví dụ ta có 1 hoặc nhiều queue chung để chứa các phần tử dữ liệu, có rất nhiều thread

cùng đồng thời thêm vào các phần tử và cũng có nhiều thread cùng đồng thời lấy ra các phần tử để đem đi xử lý. Đây là design pattern Producer-Consumer phổ biến.

### 3.5.5 BlockingQueue Interface

- BlockingQueue là một Interface kế thừa từ Queue Interface. BlockingQueue định nghĩa thêm các method để dành riêng cho việc xử lý trong môi trường multi thread khi mà nhiều thread đều muốn thêm hoặc lấy ra phần tử dữ liệu để xử lý. Sau đây là các method định nghĩa thêm của BlockingQueue.

| Method                                                 | Description                                                                                                                                                                                                                                 |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| void put(E e)                                          | Thêm một phần tử vào Queue và trả về giá trị True, nếu bị quá giới hạn dung lượng thì có thể sẽ đợi (waiting) cho tới khi có thêm dung lượng để thêm được phần tử mới.                                                                      |
| boolean offer(E e, long timeout, TimeUnit unit)        | Thêm một phần tử vào Queue và trả về giá trị True, nếu bị quá giới hạn dung lượng thì có thể sẽ đợi (waiting) cho tới khi có thêm dung lượng để thêm được phần tử mới hoặc hết thời gian đợi cho phép (timeout) và sẽ trả về giá trị false. |
| E take() throws                                        | Trả ra và đồng thời xóa bỏ phần tử đầu ra khỏi Queue. Nếu Queue rỗng thì sẽ đợi (waiting) cho tới khi nào có phần tử để lấy ra.                                                                                                             |
| E poll(long timeout, TimeUnit unit)                    | Trả ra và đồng thời xóa bỏ phần tử đầu ra khỏi Queue. Nếu Queue rỗng thì sẽ đợi (waiting) cho tới khi nào có phần tử để lấy ra hoặc trả ra Null nếu quá thời gian đợi (timeout).                                                            |
| int remainingCapacity();                               | Trả ra số lượng phần tử có thể thêm vào Queue, method này không chạy đồng bộ và ko cần phải đồng bộ.                                                                                                                                        |
| boolean remove(Object o);                              | Xóa bỏ phần tử tương ứng o ra khỏi hàng đợi và trả về giá trị true nếu hàng đợi thay đổi bởi lời gọi hàm.                                                                                                                                   |
| public boolean contains(Object o);                     | Trả ra giá trị true nếu hàng đợi chưa đựng ít nhất 1 phần tử có giá trị tương ứng với o                                                                                                                                                     |
| int drainTo(Collection<? super E> c);                  | Lấy và xóa tất cả các phần tử đang có trong queue và chuyển sang collection c. Method này trả ra số phần tử được chuyển thành công sang collection c                                                                                        |
| int drainTo(Collection<? super E> c, int maxElements); | Lấy và xóa tối đa maxElements các phần tử đang có trong queue và chuyển sang collection c. Method này trả ra số phần tử được chuyển thành công sang collection c                                                                            |

#### 3.5.5.1 LinkedBlockingQueue

- Ta xét ví dụ sau về việc khai báo, khởi tạo một Queue

```

23
24     LinkedBlockingQueue<Integer> lbqNumbers = new LinkedBlockingQueue();
25     lbqNumbers.put(1001);
26     lbqNumbers.put(1002);
27     lbqNumbers.put(1003);
28     lbqNumbers.put(1004);
29
30     while (true) {
31         System.out.println("Head of Queue " + lbqNumbers.take().toString());
32         System.out.println("-----");
33     }
34

```

- o Tại dòng 24 ta khai báo và khởi tạo một biến đối tượng lbqNumbers. Ta cũng có thể khai báo Queue với giới hạn dung lượng tối đa số lượng 1000 phần tử như sau:

- ```

23
24     LinkedBlockingQueue<Integer> lbqNumbers = new LinkedBlockingQueue(1000);

```
- o Dòng 25,26,27,28 lần lượt thêm số lượng các phần tử vào Queue;
  - o Dòng 30, 31 được thiết lập để lấy ra các phần tử tại đầu queue và in ra màn hình. Method take() sẽ lấy phần tử đầu của Queue và sẽ chờ (waiting) cho tới khi nào có phần tử mới được lấy ra. Vì vậy vòng lặp while sẽ không bị lặp liên tục cho tới khi method take() trả ra giá trị. Và kết quả là:

```

Output - JavaCoreAdvance (run) #2 ×
run:
Head of Queue 1001
-----
Head of Queue 1002
-----
Head of Queue 1003
-----
Head of Queue 1004
-----
```

- Ta nên sử dụng khi chỉ có nhu cầu thêm và lấy các phần tử ở đầu queue và không có nhu cầu truy cập ngẫu nhiên tới các phần tử ở các vị trí khác trong Queue, bởi vì Queue được thiết lập dựa trên danh sách liên kết đôi giữa các phần tử. Và tốc độ vào ra queue sẽ đạt tốc độ tối ưu nhất trong 3 loại Queue: [LinkedBlockingQueue](#), [ArrayBlockingQueue](#), [PriorityBlockingQueue](#)

### 3.5.5.2 *ArrayBlockingQueue*

- Ta nên sử dụng khi có thêm nhu cầu truy cập ngẫu nhiên tới các phần tử ở các vị trí khác trong Queue, bởi vì Queue được thiết lập dựa trên danh sách mảng các phần tử có thứ tự (ordered).

### 3.5.5.3 PriorityBlockingQueue

- Ta nên sử dụng khi muốn tạo lập một hàng chờ nhưng có sự ưu tiên giữa các phần tử dựa vào một tiêu chí so sánh. Các phần tử thêm vào **PriorityBlockingQueue** nếu muốn tính năng ưu tiên hoạt động thì phải là những loại Object có class được implement từ Comparable Interface.

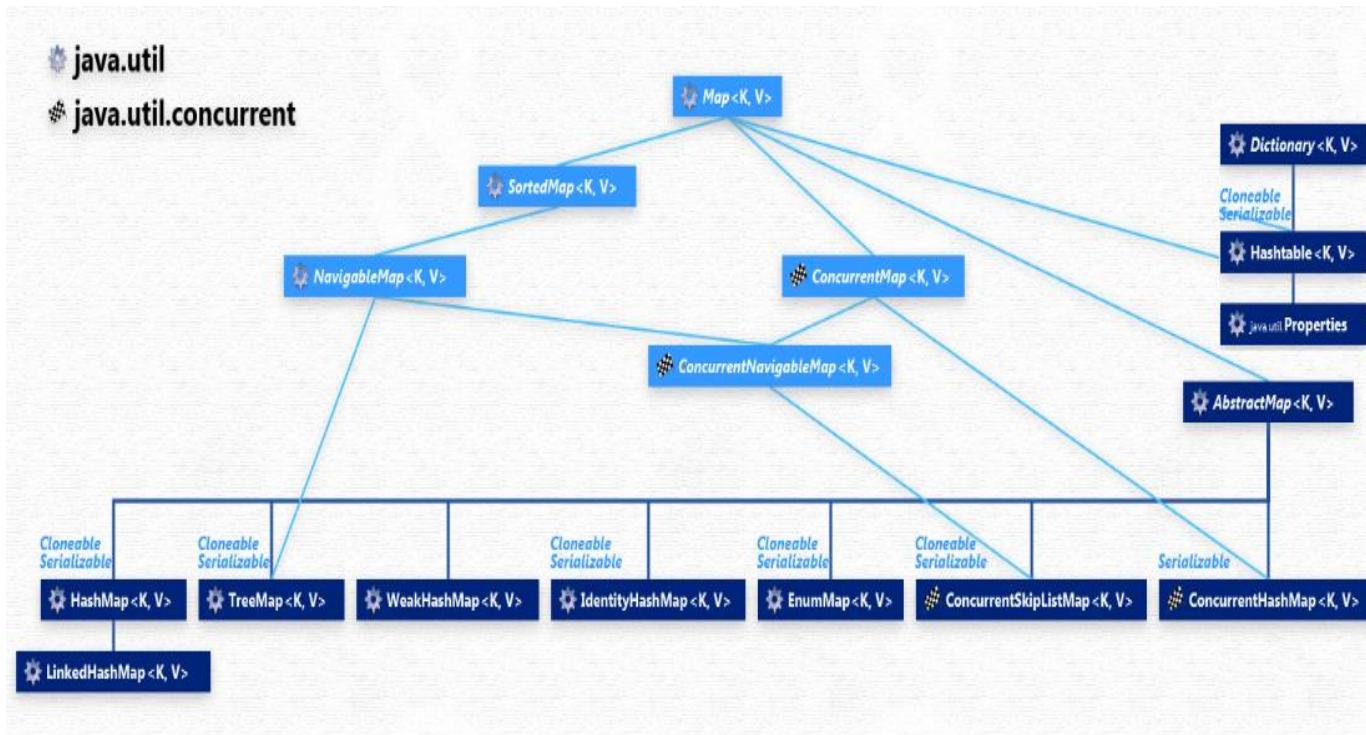
## 3.6 Map trong Java

- Map là một cấu trúc dữ liệu hoàn hảo để lưu trữ và truy xuất dữ liệu theo cặp key và value. Mỗi cặp key và value được gọi là mục nhập (entry). Mỗi Key là giá trị duy nhất trong một map. Map rất hữu ích nếu bạn phải tìm kiếm, cập nhật hoặc xóa các phần tử trên dựa vào các key.

### 3.6.1 Map Interface và các lớp thực thi của Map

- Các method quan trọng của Map Interace.

Method	Description
<code>Object put(Object key, Object value)</code>	Thêm một cặp dữ liệu key-Value vào trong map hiện tại.
<code>void putAll(Map map)</code>	Thêm các cặp dữ liệu key-Value map chỉ định vào map hiện tại.
<code>Object remove(Object key)</code>	Xóa một cặp dữ liệu key-Value có key bằng key được chỉ định.
<code>Object get(Object key)</code>	Truy cập tới giá trị (Value) thông qua key được chỉ định.
<code>boolean containsKey(Object key)</code>	Kiểm tra sự tồn tại của một key trong map.
<code>Set keySet()</code>	Trả về một tập hợp (Set) có chứa tất cả các keys của map.
<code>Set entrySet()</code>	Trả lại một tập hợp (Set) có chứa tất cả các keys và values của map.



### 3.6.1.1 HashTable

- Ta xét ví dụ sau về việc khai báo và sử dụng HashTable:

```

19
20     Hashtable<Integer, Student> hstStudents = new Hashtable<>();
21     Student s1 = new Student("Nguyen Hoang Nam");
22     Student s2 = new Student("Khuat Dang Quang");
23     Student s3 = new Student("Nguyễn Hoàng Mạnh Dũng");
24     hstStudents.put(1, s1);
25     hstStudents.put(3, s3);
26     hstStudents.put(2, s2);
27
28     for (int i = 1; i <= hstStudents.size(); i++) {
29         if (hstStudents.containsKey(i) == true) {
30             System.out.println("Student " + i + " " + hstStudents.get(i).getName());
31         }
32     }
33
  
```

- Dòng 20 khai báo và khởi tạo biến đối tượng **hstStudents**.
- Dòng 21,22,23 khởi tạo 3 đối tượng sinh viên
- Dòng 24 thêm đối tượng **s1** vào với vai trò Value và key = 1
- Dòng 25 thêm đối tượng **s3** vào với vai trò Value và key = 3
- Dòng 26 thêm đối tượng **s2** vào với vai trò Value và key = 2
- Dòng 29 ta kiểm tra key có tồn tại trong HashTable **hstStudents** không. Nếu có ta tiến hành truy cập tới giá trị (value) tương ứng với key và in thông tin Name của đối tượng **Student** ra màn hình tại dòng 30.

```
Output - JavaCoreAdvance (run) ×
run:
Student 1 Nguyen Hoang Nam
Student 2 Khuat Dang Quang
Student 3 Nguyen Hoang Manh Dung
```

- HashTable không cho phép cặp Key value nhận giá trị null. Nếu cố tình đưa vào một Key hoặc Value Null thì ta sẽ nhận được một exception lúc runtime như sau:

```
33 |     Student s4 = null;
34 |     hstStudents.put(4, s4);
35 |
Output - JavaCoreAdvance (run) ×
run:
Exception in thread "main" java.lang.NullPointerException
    at java.util.Hashtable.put(Hashtable.java:460)
    at com.activestudy.map.HashTableExample.main(HashTableExample.java:34)
```

- Không đảm bảo các cặp Key-Value được sắp xếp tuần tự (not ordered)
- Kiểu dữ liệu của Key phải implement hai method: hashCode() và equals(). Tức là các kiểu dữ liệu không nguyên thủy là được.
- HashTable Được thiết kế làm việc synchronized chính vì vậy an toàn trong môi trường Multi thread.

### 3.6.1.2 HashMap

- Ta xét ví dụ sau để thấy cách khai báo và khởi tạo một biến đối tượng HashMap

```
19 |
20     HashMap<Integer, Student> hsmStudents = new HashMap<>();
21     Student s1 = new Student("Nguyen Hoang Nam");
22     Student s2 = new Student("Khuat Dang Quang");
23     Student s3 = new Student("Nguyen Hoang Manh Dung");
24     Student s4 = null;
25     hsmStudents.put(1, s1);
26     hsmStudents.put(3, s3);
27     hsmStudents.put(2, s2);
28     hsmStudents.put(4, s4);
29     for (int i = 1; i <= hsmStudents.size(); i++) {
30         if (hsmStudents.get(i) != null) {
31             System.out.println("Student " + i + " " + hsmStudents.get(i).getName());
32         }
33     }
```

- o Dòng 20 khai báo và khởi tạo một biến đối tượng HashMap có tên hsmStudents, có key là một đối tượng Integer và Value là đối tượng kiểu Student
- o Dòng 21,22,23 khởi tạo 3 đối tượng sinh viên
- o Dòng 25 thêm đối tượng s1 vào với vai trò Value và key = 1
- o Dòng 26 thêm đối tượng s3 vào với vai trò Value và key = 3
- o Dòng 27 thêm đối tượng s2 vào với vai trò Value và key = 2

- o Dòng 28 Thêm vào một đối tượng **null**. Chú ý điểm này vì HashMap cho phép điều này.
- o Tại dòng 30. Trước khi sử dụng đối tượng (value) thì ta tiến hành kiểm tra đối tượng có null hay không để tránh **NullPointerException** xảy ra.
- o Dòng 31 xuất ra màn hình giá trị Name của các đối tượng Student trong hsmStudents.

Chúng ta thấy các đối tượng null không được xuất ra.

```
Output - JavaCoreAdvance (run) ×
run:
Student 1 Nguyen Hoang Nam
Student 2 Khuat Dang Quang
Student 3 Nguyen Hoang Manh Dung
```

- HashTable cho phép cặp Key value nhận giá trị null. Vì vậy tại dòng 28 của ví dụ, biến đối tượng s4 có giá trị null khi được đưa vào Map mà không phát sinh lỗi.
- Không đảm bảo các cặp Key-Value được sắp xếp tuần tự (not ordered)
- Kiểu dữ liệu của Key phải implement hai method: hashCode() và equals(). Tức là các kiểu dữ liệu không nguyên thủy là được.
- HashMap Không được thiết kế làm việc synchronized chính vì vậy không an toàn trong môi trường Multi thread.

### 3.6.1.3 *LinkedHashMap*

- Ta xét ví dụ sau để thấy cách khai báo và khởi tạo một biến đối tượng LinkedHashMap

```
19
20     LinkedHashMap<Integer, Student> lhsmStudents = new LinkedHashMap<>();
21     Student s1 = new Student("Nguyen Hoang Nam");
22     Student s2 = new Student("Khuat Dang Quang");
23     Student s3 = new Student("Nguyen Hoang Manh Dung");
24     Student s4 = null;
25     lhsmStudents.put(1, s1);
26     lhsmStudents.put(3, s3);
27     lhsmStudents.put(2, s2);
28     lhsmStudents.put(4, s4);
29     lhsmStudents.forEach((t, u) -> {
30         if (u != null) {
31             System.out.println("Student " + t + " " + u.getName());
32         }
33     });
34 }
```

- o Dòng 20 khai báo và khởi tạo một biến đối tượng LinkedHashMap có tên lhsmStudents có key là một đối tượng kiểu Integer và value là đối tượng kiểu Student.
- o Dòng 29 ta duyệt các phần tử theo thứ tự bằng for-each. Trong đó t là đại diện Key và u là đại diện Value. Ta thấy, khi thêm vào thế nào thì khi lấy ra cũng theo trật tự đó. Lưu ý kiểm tra value khác null trước khi sử dụng.

Output - JavaCoreAdvance (run) X

```

run:
Student 1 Nguyen Hoang Nam
Student 3 Nguyen Hoang Manh Dung
Student 2 Khuat Dang Quang

```

- Đây là class kế thừa từ HashMap.
- Các phần tử được tổ chức dưới dạng danh sách liên kết đôi, vì vậy các phần tử có thứ tự của chúng(ordered) theo Key khi mới được thêm vào danh sách.
- Chúng ta ưu tiên sử dụng LinkedHashMap nếu chúng ta cần sự sắp xếp có thứ tự mà HashMap và HashTable không có được.

#### 3.6.1.4 TreeMap

- Ta xét ví dụ sau để thấy cách khai báo và khởi tạo một biến đối tượng TreeMap.

```

19
20     TreeMap<Integer, Student> trmStudents = new TreeMap<>();
21     Student s1 = new Student("Nguyen Hoang Nam");
22     Student s2 = new Student("Khuat Dang Quang");
23     Student s3 = new Student("Nguyen Hoang Manh Dung");
24     Student s4 = null;
25     trmStudents.put(1, s1);
26     trmStudents.put(3, s3);
27     trmStudents.put(2, s2);
28     trmStudents.put(4, s4);
29     trmStudents.forEach((t, u) -> {
30         if (u != null) {
31             System.out.println("Student " + t + " " + u.getName());
32         }
33     });

```

- o Tại dòng 20 ta khai báo và khởi tạo biến đối tượng TreeMap có tên trmStudents có Key là một đối tượng Integer và value là đối tượng Student.
- o Tại dòng 28 ta thêm một đối tượng null vào và TreeMap vẫn cho phép điều này.
- o Tại dòng 29 duyệt các phần tử theo thứ tự bằng for-each. Trong đó t là đại diện Key và u là đại diện Value. Ta thấy, khi thêm vào thế nào thì khi lấy ra được sắp xếp theo thứ tự tăng dần của Key. Lưu ý kiểm tra value khác null trước khi sử dụng.

Output - JavaCoreAdvance (run) X

```

run:
Student 1 Nguyen Hoang Nam
Student 2 Khuat Dang Quang
Student 3 Nguyen Hoang Manh Dung

```

- TreeMap thực thi từ SortedMap interface.
- TreeMap đảm bảo các phần tử được sắp xếp theo thứ tự tăng dần dựa theo giá trị của Key.
- TreeMap không được thiết kế làm việc synchronized chính vì vậy không an toàn trong môi trường Multi thread.

- Nếu chúng ta cần một danh sách key value được sắp xếp theo thứ tự thì chúng ta nên sử dụng TreeMap thay vì HashMap.

### 3.7 So sánh giữa các kiểu dữ liệu Implement của Collections

- Dưới đây là bảng so sánh giữa các kiểu dữ liệu tập hợp để chúng ta dễ hình dung khi cần sử dụng chúng.

Class	Interface	Duplicate	Tuần tự (Order)	Sắp xếp (Sort)	Synchronized
ArrayList	List	No	Yes	No	No
LinkedList	List	No	Yes	No	No
Vector	List	No	Yes	No	Yes
HashSet	Set	Yes	No	No	No
LinkedHashSet	Set	Yes	Yes	No	No
TreeSet	Set	Yes	Yes	Yes	No
HashMap	Map	No	No	No	No
LinkedHashMap	Map	No	Yes	No	No
Hashtable	Map	No	No	No	Yes
TreeMap	Map	No	Yes	Yes	No
PriorityQueue	Queue	Yes	Yes	yes	No
ConcurrentLinkedQueue	Queue	Yes	Yes	No	Yes
SynchronousQueue	BlockingQueue	Yes	Yes	No	yes
ArrayBlockingQueue	BlockingQueue	Yes	Yes	No	yes
LinkedBlockingQueue	BlockingQueue	Yes	Yes	No	yes
PriorityBlockingQueue	BlockingQueue	Yes	Yes	Yes	Yes

### 3.8 Lớp thư viện Collections trong Java

- Class Collections khác với Interface Collection, và nhiều bạn do nhầm lẫn vì vậy chỉ biết tới Collection Interface mà không hề biết tới class Collections vì vậy đã bỏ qua và không biết ứng dụng của class này, điều này rất đáng tiếc.
- Class Collections cung cấp hàng loạt các method và class tĩnh để thao tác với các kiểu dữ liệu Collection và Map. Trong phạm vi cuốn sách này ta cũng không thể có điều kiện đi tìm hiểu hết các method và class mà Collections này cung cấp. Sau đây là một số method minh họa.

Collections - Navigator X

Members <empty>

- \_collections
- \_collections0
  - addAll(Collection<? super T> c, T... elements) : boolean
  - asLifoQueue(Deque<T> deque) : Queue<T>
  - binarySearch(List<? extends Comparable<? super T>> list, T key) : int
  - binarySearch(List<? extends T> list, T key, Comparator<? super T> c) : int
  - checkedCollection(Collection<E> c, Class<E> type) : Collection<E>
  - checkedList(List<E> list, Class<E> type) : List<E>
  - checkedMap(Map<K, V> m, Class<K> keyType, Class<V> valueType) : Map<K, V>
  - checkedNavigableMap(NavigableMap<K, V> m, Class<K> keyType, Class<V> valueType) : NavigableMap<K, V>
  - checkedNavigableSet(NavigableSet<E> s, Class<E> type) : NavigableSet<E>
  - checkedQueue(Queue<E> queue, Class<E> type) : Queue<E>
  - checkedSet(Set<E> s, Class<E> type) : Set<E>
  - checkedSortedMap(SortedMap<K, V> m, Class<K> keyType, Class<V> valueType) : SortedMap<K, V>
  - checkedSortedSet(SortedSet<E> s, Class<E> type) : SortedSet<E>
  - copy(List<? super T> dest, List<? extends T> src)
  - disjoint(Collection<?> c1, Collection<?> c2) : boolean
  - emptyEnumeration() : Enumeration<T>
  - emptyIterator() : Iterator<T>
  - emptyList() : List<T>
  - emptyListIterator() : ListIterator<T>
  - emptyMap() : Map<K, V>
  - emptyNavigableMap() : NavigableMap<K, V>
  - emptyNavigableSet() : NavigableSet<E>
  - emptySet() : Set<T>
  - emptySortedMap() : SortedMap<K, V>
  - emptySortedSet() : SortedSet<E>
  - enumeration(Collection<T> c) : Enumeration<T>
  - eq(Object o1, Object o2) : boolean

```

    ● min(Collection<? extends T> coll, Comparator<? super T> comp) : T
    ● nCopies(int n, T o) : List<T>
    ● newSetFromMap(Map<E, Boolean> map) : Set<E>
    ● replaceAll(List<T> list, T oldVal, T newVal) : boolean
    ● reverse(List<?> list)
    ● reverseOrder() : Comparator<T>
    ● reverseOrder(Comparator<T> cmp) : Comparator<T>
    ● rotate(List<?> list, int distance)
    ● rotate1(List<T> list, int distance)
    ● rotate2(List<?> list, int distance)
    ● shuffle(List<?> list)
    ● shuffle(List<?> list, Random rnd)
    ● singleton(T o) : Set<T>
    ● singletonIterator(E e) : Iterator<E>
    ● singletonList(T o) : List<T>
    ● singletonMap(K key, V value) : Map<K, V>
    ● singletonSpliterator(T element) : Spliterator<T>
    ● sort(List<T> list)
    ● sort(List<T> list, Comparator<? super T> c)
    ● swap(List<?> list, int i, int j)
    ● swap(Object[] arr, int i, int j)
    ● synchronizedCollection(Collection<T> c) : Collection<T>
    ● synchronizedCollection(Collection<T> c, Object mutex) : Collection<T>
    ● synchronizedList(List<T> list) : List<T>
    ● synchronizedList(List<T> list, Object mutex) : List<T>
    ● synchronizedMap(Map<K, V> m) : Map<K, V>
    ● synchronizedNavigableMap(NavigableMap<K, V> m) : NavigableMap<K, V>
    ● synchronizedNavigableSet(NavigableSet<T> s) : NavigableSet<T>
    ● synchronizedSet(Set<T> s) : Set<T>

```

- Class Collection nằm trong package java.util. Muốn sử dụng ta cần import như sau:

```

9
10 import java.util.Collections;
11

```

- Chúng ta đi tìm hiểu ví dụ đơn giản sau về việc sử dụng các method của Collections

```

18
19     ArrayList<Integer> arrNumbers = new ArrayList();
20     Collections.addAll(arrNumbers, 1, 3, 2, 4, 5, 6);
21     Collections.sort(arrNumbers);
22     int index = Collections.binarySearch(arrNumbers, 3);
23     System.out.println("Index of value is " + index);
24

```

- o Dòng 20 là ví dụ thêm cùng lúc nhiều phần tử vào một ArrayList.
- o Dòng 21 là cách sắp xếp một ArrayList
- o Dòng 22 tìm vị trí của phần tử có giá trị 3 trong ArrayList theo thuật toán tìm kiếm nhị phân.
- o Sau đây là kết quả tìm kiếm.

Output - JavaCoreAdvance (run) ×

```

run:
Index of value is 2

```

- Như vậy muốn sử dụng các method của class Collections thì ta gọi thẳng method thông qua tên class Collection như tại dòng 20,21 và 22 của ví dụ trên.
- Chúng ta không thể khai báo một đối tượng Collections bởi vì phương thức khởi dựng đã có tính được đặt thuộc tính Private. Việc khai báo một đối tượng Collections không cần thiết bởi vì mọi method của class này đều là static và có thể gọi trực tiếp thông qua tên class như đã chỉ ra ở trên.

### 3.9 Lambda Expression

- Là một hình thức thực thi một method của Interface mà không cần chỉ ra tên method.
- Ta xét ví dụ sau:

```

23 |     ArrayList<Integer> numbers = new ArrayList();
24 |     Collections.addAll(numbers, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11);
25 |     for (int i = 1; i < numbers.size(); i++) {
26 |         System.out.print(" " + i);
27 |     }
28 |

```

- o Để thực hiện công việc đơn giản trên ta mất tới 3 dòng code để in lần lượt các giá trị trong danh sách numbers.

Output - JavaCoreAdvance (run) x

```

run:
1 2 3 4 5 6 7 8 9 10 11

```

- o Bây giờ thay vì viết code như trên ta thay bằng forEach và kết hợp với biểu thức lambda như sau

```

29 |     numbers.forEach(i->{System.out.print(" " + i);});
30 |
31 |

```

- o Rất là nhanh gọn đúng không nào. Tuy nhiên nếu chúng ta chỉ dùng forEach mà không biết kết hợp với lambda thì trông nó sẽ như thế nào?

```

32 |     numbers.forEach(new Consumer<Integer>() {
33 |         @Override
34 |         public void accept(Integer t) {
35 |             System.out.print(" " + t);
36 |         }
37 |     );
38 |
39 |

```

- o Thị ra là thế này đây. `forEach()` yêu cầu tham số là một `Consumer<? super E>` và ta cần phải khai báo một instance của `Consumer`. Nhưng `Consumer` lại là một Interface, vì vậy ta



phải đi Implement lại interface này giống như ví dụ trên tại dòng 31 và override lại method `accept`.

- Rõ ràng là biểu thức lambda đã nhanh chóng định nghĩa ra một thân(body method) của method `accept` mà không cần phải chỉ ra tên method `accept` và thậm chí là tên interface `Consumer`. Điều này giúp chúng ta rút ngắn khá nhiều code thừa thãi. Và ta gọi biểu thức lambda là cách để viết một method ẩn danh (anonymous method).
- Phía trên là ta đề cập tới tính tiện dụng và khả năng mà nó định nghĩa thay thế cho một interface. Vậy giờ ta xét sâu hơn hình thái mà nó thay thế cho Interface như thế nào nhé.

- Xét vong lặp `forEach` yêu cầu truyền vào một Consumer Interface.

```
public void forEach(Consumer<? super E> action) {
```

- Consumer Interface có hình thái như sau:

```
41  @FunctionalInterface
42  public interface Consumer<T> {
43
44      /**
45      * Performs this operation on the given argument.
46      *
47      * @param t the input argument
48      */
49
50      void accept(T t);

```

- Biểu thức Lambda có thể viết bằng một trong hai cách như sau đều không có sự khác biệt:

```
29      numbers.forEach(i->{System.out.print(" " + i);});
30
31
29      numbers.forEach((Integer i)->{System.out.print(" " + i);});
30
31
```

Tách riêng biểu thức Lambda như sau:

```
(Integer i)->{System.out.print(" " + i)}
```

- ⇒ Ta nhận thấy (`Integer i`) chính là tham số truyền vào của phương thức `accept(T t)`;
- ⇒ `{System.out.print(" " + i);}` Chính là phần body của phương thức `accept(T t)`;
- ⇒ Dấu `->` Chính là phần ngăn cách giữa tham số truyền vào và body(Nội dung thực thi).

- Nội dung thực thi của Lamda expression có thể là 1 khối lệnh hoặc 1 biểu thức. Dấu “->” tách biệt các tham số và nội dung thực thi (Body)

- Biểu thức lambda thực thi được đổi với những interface có một method hoặc nếu interface có nhiều method thì tồn tại một method chưa được define và các method khác đã được define default.
- Nhằm làm giảm thiểu số dòng code
- Biểu thức lambda được áp dụng cho những method cho phép truyền vào một interface và sau đây là ví dụ về trường hợp sử dụng biểu thức Lambda:

```

52
53     Collections.sort(numbers, (Integer c, Integer d) -> {
54         return c > d ? 1 : (c < d ? -1 : 0);
55     );
56

```

Ví dụ trên biểu thức Lambda kết hợp với sort trong Collections để implement Interface `Comparator<T>` và define method `int compare(T o1, T o2);`

```

63
64     System.out.println("+" + numbers.stream().filter(i -> i % 2 != 0)
65             .filter(i -> i > 3)
66             .filter(i -> i < 11)
67             .max((Integer c, Integer d) -> c.compareTo(d))
68             .map(i -> i * i)
69             .get());

```

Ví dụ trên biểu thức Lambda kết hợp với Stream API trong method filter để implement Interface `Predicate<? super T>` và define method `boolean test(T t);`

- Trong phần tiếp theo chúng ta nghiên cứu sâu về Stream API và sẽ thấy rõ hơn sự kết hợp giữa biểu thức Lambda và Stream API.

### 3.10 Stream API trong Java

- Stream API trong bộ Collection API Được giới thiệu lần đầu ở bản Java 8 (JDK 1.8).
- Stream API là cách thức mới để thao tác với các kiểu dữ liệu Collection, bên cạnh các cách thức thông thường như vòng lặp for và bộ lặp iterator thì Stream cũng cung cấp khả năng duyệt và thao tác với từng phần tử.
- Hiểu theo cách khác khi sử dụng Stream API bạn có thể xử lý dữ liệu dạng Collection, Map 1 cách tự nhiên giống như các câu lệnh SQL. Ví dụ ta có câu SQL sau:

`SELECT SUM(salary) FROM Employee WHERE YearOfBirth > 1990;`

- Stream là một Interface nằm trong Package `java.util.stream` muốn sử dụng thì chúng ta import như sau:

```

10
11     import java.util.stream.Stream;
12

```

Hoặc sử dụng thông qua Collection, Map đã Implement Stream Interface...

- Ta đi xét ví dụ sau:

```

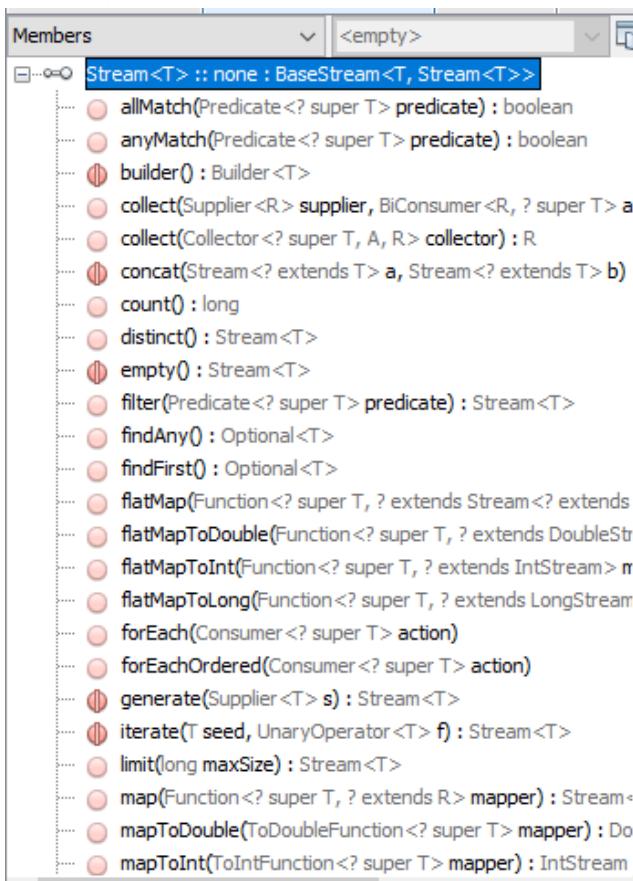
19      ArrayList<Integer> arrNumbers = new ArrayList();
20      Collections.addAll(arrNumbers, 1, 3, 2, 4, 5, 6, 7, 8, 9, 10);
21      int max = arrNumbers.stream()
22          .filter(i->i>3)
23          .filter(i->i<10)
24          .max((x,y)->x.compareTo(y))
25          .get();
26
27
28      System.out.println("Max: " + max);

```

- o Dòng số 22 chúng ta truy cập tới Stream được tạo ra bởi nguồn từ arrNumbers
- o Dòng số 23 gọi method filter và truyền vào một biểu thức Lambda để đặt điều kiện lọc cho stream, ý nghĩa là lọc những phần tử có giá trị >3
- o Dòng số 24 cũng tương tự như dòng 23 ý nghĩa là lọc những phần tử có giá trị <10
- o Dòng số 25 tìm ra giá trị Max sau khi đã filter ở dòng 23 và 24. Truyền vào biểu thức Lambda để chỉ ra cách so sánh giữa các phần tử.
- o Dòng số 26 lấy ra và trả về giá trị đã thực hiện ở các bước dòng 23 đến 25 và gán cho biến max.
- o Dòng số 28 hiển thị ra giá trị max vừa tìm được. Và sau đây là kết quả

The screenshot shows the 'Output' window of a Java IDE. It contains two lines of text: 'run:' and 'Max: 9'. The 'run:' line is preceded by a green play button icon. The 'Max: 9' line is preceded by a yellow play button icon.

- Stream Interface có rất nhiều method để ta vận dụng.



### 3.10.1 Stream Song song (Parallel Stream)

- Các stream có thể được chạy tuần tự (sequential) hoặc song song (parallel). Các thao tác trên các stream tuần tự được thực hiện trên một luồng đơn (single thread) trong khi các phép toán trên các stream song song được thực hiện đồng thời trên nhiều luồng (multi-thread).
- Chúng ta thường sử dụng Parallel Streams trong môi trường multi-thread khi mà chúng ta cần hiệu suất xử lý nhanh.
- Chúng ta đi xét ví dụ sau:

```

67
68     ArrayList<Integer> arrNumbers = new ArrayList();
69     Collections.addAll(arrNumbers, 1, 3, 2, 4, 5, 6, 7, 8, 9, 10);
70     arrNumbers.parallelStream()
71         .forEach(i -> System.out.print(" " + i));

```

- o Thay vì sử dụng Stream chúng ta đơn giản là sử dụng parallelStream(). Nhưng khi dùng Stream này thì quá trình thực thi không còn giữ được trình tự xử lý giữa các phần tử nữa và đã có sự xáo trộn về trình tự thực hiện như sau:

```
Output - JavaCoreAdvance (run) X
run:
7 2 9 5 6 4 10 3 1 8
```

### 3.10.2 Các cách tạo Stream:

#### 3.10.2.1 Tạo stream từ collection

- Ta có thể tạo ra Stream từ các loại dữ liệu Collection, Map.

```
20 | ArrayList<Integer> arrNumbers = new ArrayList();
21 | Collections.addAll(arrNumbers, 1, 3, 2, 4, 5, 6, 7, 8, 9, 10);
22 | int max = arrNumbers.stream()
23 |     .filter(i->i>3)
24 |     .filter(i->i<10)
25 |     .max((x,y)->x.compareTo(y))
26 |     .get();
```

#### 3.10.2.2 Array

- Chúng ta có thể tạo ra Stream từ Array.

```
33 | String[] names = {"Nam", "Quang", "Dũng", "Thái"};
34 | stream.of(names).forEach(i->System.out.println(i));
35 |
```

#### 3.10.2.3 empty stream

- empty Stream dùng để khởi tạo một Stream, bởi vì Stream tạo ra trống rỗng ko có dữ liệu.

```
36 |
37 | Stream<String> streamEmpty = Stream.empty();
38 |
```

#### 3.10.2.4 Stream.builder()

- StreamBuilder được dùng để thêm vào các phần tử vào Stream.

```
39 |
40 | Stream<String> streamBuilder
41 |     = Stream.<String>builder().add("Nam").add("Quang").build();
42 |
```

#### 3.10.2.5 Stream.generate()

- Tạo ra Stream với số lượng lớn các phần tử mang cùng các giá trị số lượng phần tử nên được giới hạn bởi method limit.

```

42 |     Stream<String> streamGenerated
43 |             = Stream.generate(() -> "Active Study").limit(15);
44 |
45 |

```

### 3.10.2.6 Stream.iterate()

- Tạo ra Stream với số lượng lớn các phần tử có thể khác nhau. số lượng phần tử nên được giới hạn bởi method limit.

```

46 |     Stream<String> streamIterated
47 |             = Stream.iterate("Active Study", n -> n + 1).limit(5);
48 |
49 |     List<String> listIterated = streamIterated.collect(Collectors.toList());
50 |     listIterated.forEach(i -> System.out.print(i + "; "));
51 |

```

### 3.10.2.7 Stream of Primitives

- Stream tạo ra từ int;

```

52 |
53 |     IntStream intStream = IntStream.range(1,10);
54 |

```

### 3.10.2.8 Stream of File

- Stream được tạo từ nội dung file

```

62 |
63 |     Path path = Paths.get("D:\\temp\\\\example.log");
64 |     Stream<String> streamOfStrings = Files.lines(path);
65 |     Stream<String> streamWithCharset
66 |             = Files.lines(path, Charset.forName("UTF-8"));
67 |     streamWithCharset.forEach(i-> System.out.println(i));

```

## 3.11 Generic

- Ta xét ví dụ sau về việc tạo một ArrayList và chỉ cho phép một kiểu giá trị Integer được phép thêm vào danh sách.

```

ArrayList<Integer> mylist = new
    ArrayList<Integer>();
mylist.add(10);
mylist.add("Hi");//error
mylist.add(true);//error
mylist.add(15);

```

## Generic

- Chúng ta đã sử dụng generic với các kiểu dữ liệu Collection và Map để tạo ra tập hợp dữ liệu chỉ chứa một loại dữ liệu, việc này giúp cho chương trình của chúng ta sáng sủa hơn và tránh được các lỗi logic về dữ liệu.

- o Chúng ta thử hình dung nếu không dùng Generic thì sẽ ra sao nếu chúng ta Add vào danh sách một đối tượng kiểu String và sau đó loay hoay lấy ra như một kiểu int như ví dụ sau

```

16 |   ArrayList myList = new ArrayList();
17 |   myList.add("1");
18 |   myList.add("2");
19 |   myList.add("3");
20 |   myList.add("Bốn");
21 |   myList.add("5");
22 |   int value = 0;
23 |   for(Object i: myList){
24 |       value = (int)i;
25 |   }

```

- o Sẽ chẳng xảy ra vấn đề gì khi không có dòng 20 và kết quả là chúng ta nhận được một RuntimeException với nội dung đại loại như: Không thể chuyển đổi một kiểu String sang Integer. Và rõ ràng là chúng ta khó lòng kiểm soát hết được việc dữ liệu của chúng ta có được add vào như tại dòng 20 hay không. Đây đã thực sự là một vấn đề đối với lập trình viên ở Java phiên bản 5.

Output - JavaCoreAdvance (run) |

in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer  
at com.activestudy.collection.GenericExample.main(GenericExample.java:24)

- o Vậy khi dùng Generic thì chúng ta đã tránh được việc thêm vào danh sách một phần tử có kiểu dữ liệu không mong muốn theo mô tả và khi lấy phần tử ra ta cũng không cần phải chuyển đổi từ Object sang kiểu dữ liệu của chúng ta nữa vì trình biên dịch tự hiểu được điều này. Bây giờ code của chúng ta sẽ trở thành như sau

```

16      ArrayList<Integer> myList = new ArrayList();
17      myList.add(1);
18      myList.add(2);
19      myList.add(3);
20      myList.add("Bốn");
21      myList.add(5);
22      int value = 0;
23      for(int i: myList){
24          value = i;
25      }

```

- o Tại dòng 20 chúng ta nhận ra ngay phần tử ngoại đao đã vô tình được thêm vào và chúng ta sẽ nhanh chóng sửa chữa code ngay từ khi lập trình.
- o Tại dòng 24 chúng ta không cần phải chuyển đổi kiểu dữ liệu nữa, điều này làm cho chương trình của chúng ta chạy nhanh hơn và không phát sinh các ngoại lệ\
- Chúng ta có thể tự tạo ra một class sử dụng Generic riêng của chúng ta như sau:

```

12      * @author tanth
13
14      public class MyGenericExample<T> {
15          ArrayList<T> objects = new ArrayList<>();
16          public void add(T t) {
17              objects.add(t);
18          }
19          public T get(int index){
20              return objects.get(index);
21          }
22      }

```

### 3.12 Đa luồng (Multi Thread) trong Java

#### 3.12.1 Tư duy lập trình đa luồng trong Java

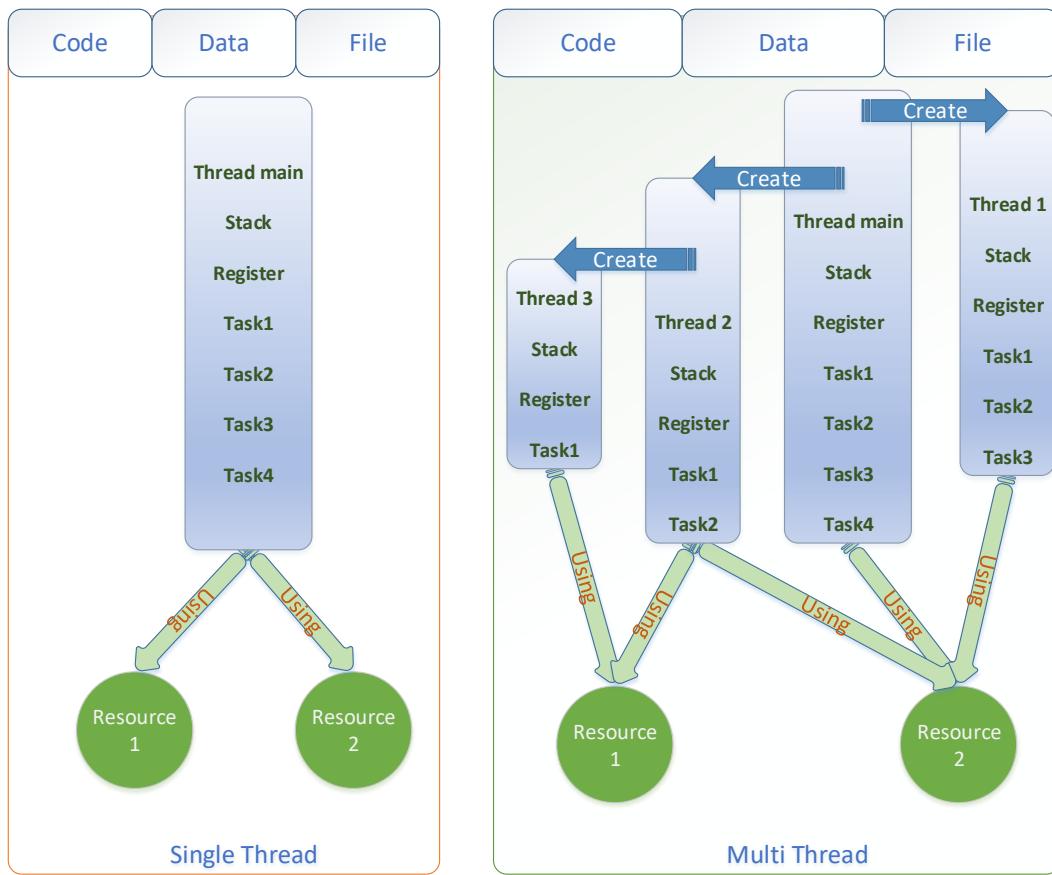
- Tư duy lập trình đơn luồng là tư duy theo lối lập trình tuần tự. Các khối lệnh được thực hiện nối tiếp nhau, lệnh trước thực hiện xong thì sẽ thực hiện tiếp các lệnh sau theo đúng như trình tự mà chúng ta đã sắp xếp khi lập trình.
- Trong tư duy lập trình đa luồng thì chúng ta cần phải hiểu rằng các khối lệnh của chúng ta không còn được đảm bảo chạy tuần tự như lập trình đơn luồng nữa, các khối lệnh sẽ được đồng thời thực hiện bởi nhiều luồng (thread), các dữ liệu cũng được xử lý trên nhiều luồng (thread), chính điều này tạo nên sự xáo trộn trong suy nghĩ của chúng ta về tính logic trong xử lý đa luồng so với đơn luồng. Giờ đây tính logic về việc sử dụng dữ liệu tuần tự bị phá vỡ, tính logic về sử dụng tài nguyên tuần tự bị phá vỡ, tính logic về việc các đoạn mã lệnh được thực hiện tuần tự bị phá vỡ.

Chúng ta phải chấp nhận một thực tại rằng đối với lập trình đa luồng mọi thứ đều có thể không kiểm soát được trình tự.

- Trong lập trình đa luồng, lập trình viên lúc này như một nhà quản lý. Hãy hình dung chúng ta có bao nhiêu thread thì tương ứng với bấy nhiêu con người cần được chúng ta điều phối sắp xếp để làm việc chung được với nhau, phối hợp được với nhau thành một trình tự logic đúng và an toàn như một công ty hay tổ chức. Ở công ty và tổ chúng của chúng ta nơi mà sẽ phải đảm bảo các yếu tố:
  - o Mọi người đều có việc làm
  - o Mọi người có thể trao đổi dữ liệu cho nhau mà không gặp rắc rối gì
  - o Mọi người đều có thể phối hợp công việc với nhau mà không phải lãng phí thời gian chờ đợi nhau.
  - o Mọi người có thể sử dụng chung tài nguyên như chỗ ngồi, điều hòa, máy in, máy scan... mà không gặp bất kỳ xung đột gì
- Đối với lập trình đa luồng cần phải đảm bảo được các yêu tố
  - o Hiệu suất chương trình cao
  - o Đảm bảo đúng trình tự logic
  - o Không bị bế tắc, tức là không xuất hiện các deadlock
- Lập trình đa luồng cần phải có được các kỹ thuật sau:
  - o Thiết kế, quy hoạch và tạo được các luồng xử lý công việc
  - o Tổ chức phân phối dữ liệu
  - o Kỹ thuật đồng bộ Synchronization
  - o Kỹ thuật tránh deadlock

### 3.12.2 Luồng (Thread) là gì

- Mỗi một chương trình đều có ít nhất một thread để chạy ta gọi là thread main.
- Một thread liên tục thực hiện các công việc của nó theo một trình tự.
- Trong môi trường multithread chúng ta có thể tạo nhiều thread trong một chương trình để cùng đồng thời thực hiện cùng một hoặc nhiều tác vụ khác nhau.
- Trong cùng một chương trình các Thread cần được liên tục trao đổi dữ liệu và dùng chung tài nguyên với nhau.



- Một Thread sẽ kết thúc khi đã thực hiện xong tất cả các tác vụ của nó hoặc Thread main dừng.
- Có hai cách để tạo ra một thread trong java như sau:
  - o Extend the `java.lang.Thread` class
  - o Implement the `Runnable` interface

Và sau đây chúng ta sẽ nghiên cứu từng cách để vận dụng.

### 3.12.3 Creating a Thread Using the Thread Class

- Chúng ta xét ví dụ sau để xem cách tự tạo một thread thông qua class.

```

10  * @author tanth
11  */
12  public class ThreadExample {
13      public static void main(String[] args) {
14          MyThread myThread = new MyThread();
15          myThread.start();
16      }
17  }
18  class MyThread extends Thread {
19      @Override
20      public void run() {
21          int i = 0;
22          while (i < 100000) {
23              System.out.println("I am running....." + i++);
24          }
25          System.out.println("I am shutting down, Goodbye");
26      }
27  }

```

- o Xét ví dụ trên ta đã thực hiện các bước sau:

- Bước 1: Tại dòng 18 chúng ta khai báo class MyThread được extends từ class Thread của Java. Class MyThread tiến hành Override lại phương thức run() và thực hiện các nhiệm vụ của nó trong method run() này.
- Bước 2: Tại dòng 14 chúng ta khai báo một biến đối tượng myThread do chúng ta tự tạo và khởi tạo đối tượng này.
- Bước 3: Tại dòng 15 chúng ta bắt đầu khởi chạy đối tượng Thread của chúng ta.
- o Tại dòng 22 của method run() thực hiện lặp lại công việc trong khi  $i < 100000$ . Khi vòng lặp while này dừng lại cũng là lúc Thread này chuẩn bị kết thúc công việc
- o Tại dòng 25 là công việc cuối cùng của Thread và sau đó Thread do chúng ta tạo ra sẽ dừng lại.
- o Kết quả của việc chạy Thread như sau:

The screenshot shows the 'Output' window of a Java IDE during the execution of the 'run()' method. The window title is 'Output - JavaCoreAdvance (run)'. The output text is as follows:

```
I am running.....99992
I am running.....99993
I am running.....99994
I am running.....99995
I am running.....99996
I am running.....99997
I am running.....99998
I am running.....99999
I am shutting down, Goodbye
```

### 3.12.4 Creating a Thread Using the Runnable Interface

- Chúng ta xét ví dụ sau để xem cách tự tạo một thread thông qua Runnable Interface và Thread class.

```

10  * @author tanht
11  */
12  public class ThreadImplementExample {
13      public static void main(String[] args) {
14          MyImplementThread myImplementThread = new MyImplementThread(),
15          Thread myThread = new Thread(myImplementThread);
16          myThread.start();
17      }
18  }
19
20  class MyImplementThread implements Runnable {
21      @Override
22      public void run() {
23          int i = 0;
24          while (i < 100000) {
25              System.out.println("I am running....." + i++);
26          }
27          System.out.println("I am shutting down, Goodbye");
28      }
29  }

```

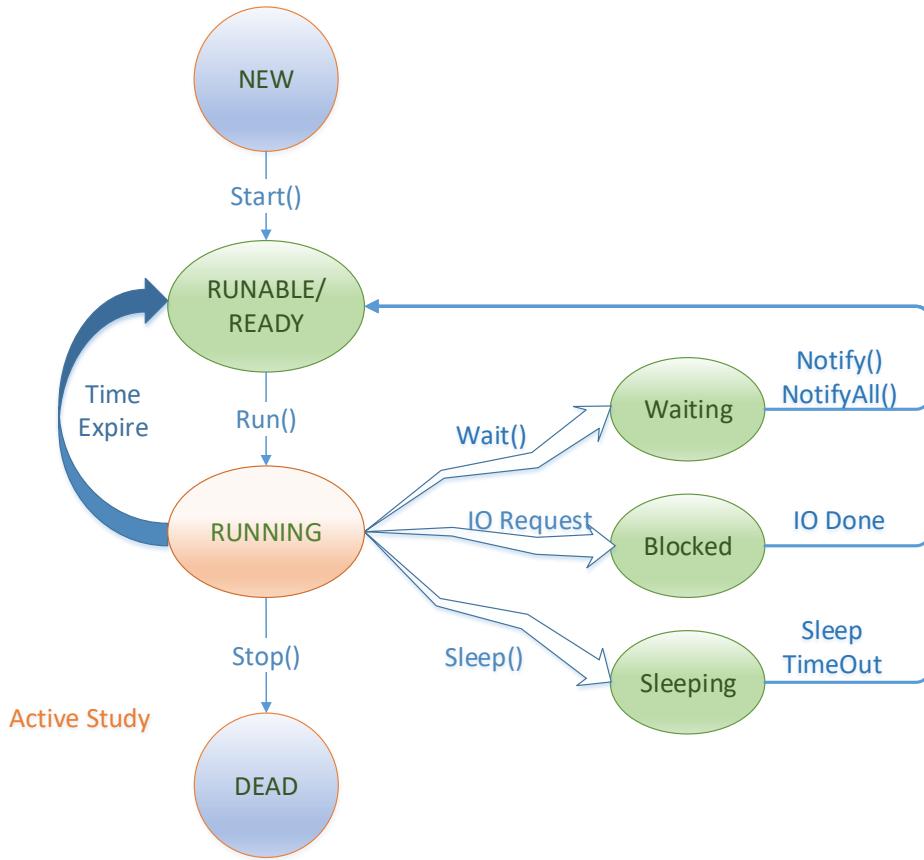
- o Xét ví dụ trên ta nhận thấy để tạo và chạy được thread cần thực hiện các bước sau:
  - Bước 1: Tại dòng 20 chúng ta tạo Class MyImplementThread và Implement Runnable Interface.
  - Bước 2: Tại dòng 14 khai báo biến đối tượng myImplementThread và khởi tạo đối tượng
  - Bước 3: Tại dòng 15 khai báo biến đối tượng myThread và khởi tạo đối tượng Thread bằng cách truyền biến đối tượng myImplementThread vào phương thức khởi dụng.
  - Bước 4: Start Thread bằng cách gọi method Start() như tại dòng 16.
- o Tại dòng 22 method run() được Override và thực hiện các tác vụ của Thread. Method này dừng lại thì thread cũng dừng lại theo.
- o Sau đây là kết quả chương trình, có thể nói là không khác gì so với cách chạy thread bằng cách extend từ class Thread.

The screenshot shows the Java IDE's Output window titled "Output - JavaCoreAdvance (run)". It displays the following text output:

```
I am running.....99995
I am running.....99996
I am running.....99997
I am running.....99998
I am running.....99999
I am shutting down, Goodbye
```

### 3.12.5 Vòng đời của Thread (Lifecycle of a Thread)

- Phần này chúng ta sẽ đi nghiên cứu về các trạng thái trong suốt vòng đời hoạt động của một thread.
- Trong java các trạng thái của thread thật sự là phong phú và linh hoạt, nó có tới 7 trạng thái. Trong suốt quãng đời tồn tại và hoạt động nó chuyển đổi qua lại giữa các trạng thái này nhiều lần mà thậm chí lập trình viên còn không hề hay biết. Sau đây là cụ thể vòng đời của Thread như sau:



- o Kể từ khi sinh ra bằng cách khởi tạo một biến đối tượng của lớp thread, thì thread này bắt đầu ở trạng thái **New**.
- o Sau Khi phương thức `start()` của thread được gọi tới, nhiều bạn nhầm tưởng rằng sau khi gọi phương thức này thì thread đã bắt đầu được chạy ngay lập tức, thực sự là không phải thế. Lúc này thread mới đang ở trạng thái **runable** hay còn gọi là **ready**, tức là trạng thái sẵn sàng để bộ lập lịch (Scheduler) sắp lịch để chờ được chạy.
- o Và vào một thời điểm tiếp theo đó thì thread này mới được chiếm tài nguyên CPU và sẽ thực sự được chạy, tức là thread này chuyển sang trạng thái **running**. Khi ở trạng thái **running** thì phương thức `run()` của thread được triệu gọi và thực thi các công việc của nó ở đây.

- Những khoảng thời gian sau đó nếu thực thi xong các lệnh ở trong `run()` và thoát ra khỏi phương thức `run()` thì thread sẽ quay về trạng thái **Dead**, tức là kết thúc vòng đời của một thread.
  - Tuy nhiên, khi ở trạng thái **running** thread nhiều lần được chuyển qua lại các trạng thái **runnable**, **sleeping**, **blocked**, **waiting** nữa trước khi chuyển lại về trạng thái **running**.
  - Các trạng thái **Waiting**, **Blocked**, **Sleeping** được gọi chung là trạng thái **Nonrunnable**: Là các trạng thái tạm dừng và chưa đủ điều kiện về logic để chạy. Thread muốn được chạy thì phải thỏa mãn một điều kiện nào đó (tùy từng trạng thái) và chuyển sang trạng thái ready mới có thể tiếp tục được chạy.
- Sau đây là các trạng thái trong vòng đời của thread:

#### 3.12.5.1 Trạng thái New của Thread

- Đây là trạng thái đầu tiên của thread khi nó bắt đầu được khởi tạo (Khởi tạo bằng lệnh `new()` ấy). Và ở trạng thái này thread chưa được chạy, cũng chưa được lập lịch để chạy.

#### 3.12.5.2 Trạng thái Ready/runnable của Thread

- Đây là trạng thái mà thread sẵn sàng và chờ được chạy vì nó đang được lập lịch để chạy. Thread sẽ mang trạng thái này khi mà lần đầu tiên phương thức `start()` của nó được triệu gọi. và nó sẽ mang trạng thái này nhiều lần nữa khi chuyển từ các trạng thái **running**, **waiting**, **sleeping**, **blocking** sang.

#### 3.12.5.3 Trạng thái Running

- Đây là trạng thái mà lập trình viên luôn mong muốn khi bắt đầu ra lệnh `start()` với nó. Ở trạng thái này các lệnh ở trong phương thức `run()` của thread được triệu gọi. Và thread chỉ có trạng thái này khi trước đó nó ở trạng thái ready.

#### 3.12.5.4 Trạng thái Blocked:

- Đây là trạng thái mà thread phải đợi một tài nguyên (resource) của hệ thống hoặc đợi một đối tượng nhả khóa ra (Object's lock) để có thể chuyển sang trạng thái ready để tiếp tục chạy. Thông thường trong java khi truy cập vào một tài nguyên hệ thống thì thread tự động bị block cho tới khi chiếm được quyền truy cập.
- Sau đây ta xét tới một ví dụ truy cập tới tài nguyên socket stream như sau:

```

27
28     class MyThreadBlock extends Thread {
29         @Override
30         public void run() {
31             Socket clientSock;
32             try {
33                 clientSock = new Socket("127.0.0.1", 10000);
34
35                 InputStream inSocketStream = clientSock.getInputStream();
36                 int len = 0;
37                 BufferedOutputStream outFile =
38                     new BufferedOutputStream(new FileOutputStream("response.txt"));
39                 byte buf[] = new byte[256];
40                 while ((len = inSocketStream.read(buf)) != -1) {
41                     outFile.write(buf, 0, len);
42                 }
43             } catch (IOException ex) {
44                 Logger.getLogger(MyThreadBlock.class.getName()).log(Level.SEVERE, null, ex);
45             }
46         }
47     }

```

- o Trong đoạn code trên tại dòng 33 ta đã khởi tạo một kết nối tới một máy có IP: 127.0.0.1 ở port 10000. Một đối tượng của InputStream có tên inSocketStream được tạo ra để đọc dữ liệu đầu vào của socket.
- o Chúng ta xét tới dòng số 40, một vòng while được đưa vào để đọc dữ liệu từ inSocketStream. Vòng while này thoát nhìn thật nguy hiểm vì nó có thể làm treo máy khi nó chạy liên tục thế này mà không có đoạn nào ngừng nghỉ hết thế này, nó chiếm hết tài nguyên cpu của hệ thống mất. Nhưng thật may mắn là không phải vậy. Vòng while chỉ chạy tiếp và chạy liên tục khi có dữ liệu từ đầu của máy 127.0.0.1 truyền tới socket mà thôi. Thật vậy và nó sẽ dừng lại ở lệnh `inSocketStream.read(buf)` khi không có dữ liệu truyền tới. và cụ thể là nó đang ở trạng thái **blocked**.
- Sau khi có dữ liệu tiếp tục được truyền tới socket, stream `inSocketStream` lại có dữ liệu thì thread lại được chuyển sang trạng thái **ready** để tiếp tục được sắp lịch để lấy dữ liệu trong stream ra `buf`.
- Vậy là ta đã quan sát được quá trình một thread đang ở trạng thái **running** chuyển sang **Blocked** sau đó về **ready** và lại **running** khi thỏa mãn điều kiện Stream có data tới.
- Đối với một Object lock ta sẽ xét cụ thể ví dụ ở phần sau liên quan tới “Synchronization and Locks”

### 3.12.5.5 Trạng thái Sleeping:

- Đây là trạng thái ngủ của thread xảy ra khi ta gọi phương thức `sleep()` của thread. đây là phương thức static có 2 overload có dạng như sau:

301  
325

```
public static native void sleep(long millis)
public static void sleep(long millis, int nanos)
```

- Thread sẽ ở trạng thái Sleeping trong thời gian milliseconds (phần nghìn của giây, 1 giây = 1000 milliseconds) hoặc ngủ trong thời gian milliseconds và nanoseconds (Phần triệu của giây 1 giây = 1000000 nanoseconds), sau đó Thread tự động chuyển sang trạng thái ready để sẵn sàng chạy tiếp.
- Mục đích khi sử dụng phương thức `sleep()` này của thread là trong các tình huống lập trình viên chủ ý muốn tạo ra một bộ định thời (scheduler) để thỉnh thoảng thread làm một Task xong là lại sleeping để đỡ tốn tài nguyên CPU như ví dụ dưới đây tại dòng 28.

```
21 class MyThread extends Thread {
22     @Override
23     public void run() {
24         int i = 0;
25         while (i < 10000) {
26             try {
27                 System.out.println("I am running....." + i++);
28                 sleep(1000);
29             } catch (InterruptedException ex) {
30                 Logger.getLogger(MyThread.class.getName()).log(Level.SEVERE, null, ex);
31             }
32         }
33         System.out.println("I am shutting down, Goodbye");
34     }
35 }
```

- o Ví dụ trên là Thread in ra màn hình dòng chữ, Nếu không có `sleep(1000)` thì chương trình liên tục in ra màn hình. Khi có `sleep(1000)` thì thread sẽ chuyển sang trạng thái sleeping trong khoảng 1 giây sau đó tiếp tục in ra màn hình.

### 3.12.5.6 Trạng thái Waiting :

- Đây là trạng thái mà một object trong thread đang chạy triệu hồi phương thức `wait()`. Thread ngay lập tức chuyển sang `waiting` và chỉ được chuyển sang trạng thái `ready` khi có một thread khác gọi tới phương thức `notify()` (của `object` đã gọi `wait()`) hoặc `notifyall()`. Ta sẽ nghiên cứu kỹ phần `wait()` và `notify()`, `notifyall()` này ở một phần tiếp theo bởi vì đây là kỹ thuật tương đối khó.
- Lý do mà ta phải đưa thread sang trạng thái `waiting` là vì chúng ta cần một thread khác phải thực hiện xong một việc của nó trước và thread hiện tại mới bắt đầu thực hiện tiếp công việc. Ví dụ như nhiều thread cùng đọc và ghi vào một đối tượng thì ta cần chờ cho các thread khác ghi vào

đối tượng xong thì mới đọc, không nên cùng đọc và ghi xảy ra cùng một lúc sẽ dẫn tới sai lệch về mặt dữ liệu. hoặc thậm chí xảy ra deadlock (ta sẽ nghiên cứu deadlock sau)

- Ngoài method `wait()` ta có thêm 3 phương thức của thread để đưa thread vào trạng thái waiting này.

```

1325  ┌─────────┐
1241  │   public final void join() { }
1291  │   public final synchronized void join(long millis)
1291  │   public final synchronized void join(long millis, int nanos)

```

- `join():`đợi mãi mãi
- `void join(long millis):` đợi millisec. sau khi hết thời gian timeout thì thread tiếp tục chuyển sang trạng thái ready để chuẩn bị chạy.
- `join(long millis, int nanosec):` đợi millisec và thêm nanosec . sau khi hết thời gian timeout thì thread tiếp tục chuyển sang trạng thái ready để chuẩn bị chạy.
- Cả 3 method `join` sẽ đều kết thúc trạng thái `waiting` để chuyển sang trạng thái tiếp theo nếu thread chưa method `join` chuyển sang trạng thái `dead` (kết thúc). Đây là điểm khác biệt của nó so với method `sleep()`
- Ta xét ví dụ sau khi sử dụng `Join()`

```

16
17  ┌─────────┐
18  │   public static void main(String[] args) {
19  │       MySleepingThread myThread1 = new MySleepingThread("Thread 1");
20  │       MySleepingThread myThread2 = new MySleepingThread("Thread 2");
21  │       MySleepingThread myThread3 = new MySleepingThread("Thread 3");
22  │       try {
23  │           myThread1.start();
24  │           myThread1.join();
25  │           myThread2.start();
26  │           myThread2.join();
27  │           myThread3.start();
28  │           myThread3.join();
29  │           System.out.println("Thread main is shutting down");
30  │       } catch (InterruptedException ex) {
31  │           //Process Exception here
32  │       }
33  │
}

```

```

35     class MySleepingThread extends Thread {
36
37     MySleepingThread(String name) {
38         super(name);
39     }
40
41     @Override
42     public void run() {
43         int i = 0;
44         while (i < 3) {
45             try {
46                 System.out.println(this.getName() + " running....." + i++);
47                 sleep(1000);
48             } catch (InterruptedException ex) {
49             }
50         }
51         System.out.println(this.getName() + " shutting down, Goodbye");
52     }
}

```

- Tại dòng 23 Thread main sẽ đợi cho tới khi nào Thread 1 kết thúc thì mới tiếp tục start Thread 2 tại dòng 24
- Tại dòng 25 Thread main cũng sẽ đợi cho tới khi nào Thread 2 kết thúc thì mới tiếp tục start Thread 3 tại dòng 26
- Tại dòng 27 Thread main cũng sẽ đợi cho tới khi nào Thread 3 kết thúc thì mới tiếp tục chạy tới dòng 28 để in thông báo ra màn hình. Và kết quả chạy chương trình như sau:

Output - JavaCoreAdvance (run)

```

run:
Thread 1 running.....0
Thread 1 running.....1
Thread 1 running.....2
Thread 1 shutting down, Goodbye
Thread 2 running.....0
Thread 2 running.....1
Thread 2 running.....2
Thread 2 shutting down, Goodbye
Thread 3 running.....0
Thread 3 running.....1
Thread 3 running.....2
Thread 3 shutting down, Goodbye
Thread main is shutting down

```

- Tại dòng 25 và 27 nếu thay method join() bằng sleep(10000) thì chúng ta sẽ thấy sự khác biệt, hãy tự tay trải nghiệm nhé.

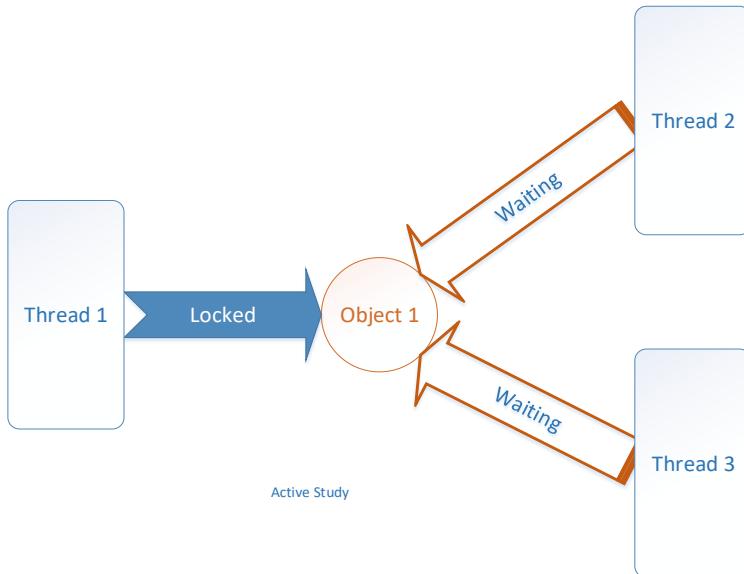
### 3.12.5.7 Trạng thái Dead

- Khi một thread đã kết thúc phương thức run() thì nó sẽ chuyển về trạng thái dead. Ở trạng thái này thread không thể khởi chạy lại một lần nữa, nếu ta cố gọi lại phương thức start() của nó ta sẽ nhận lại một exception, nhưng bản thân object này nó vẫn là một đối tượng đã được khởi tạo và chưa giải phóng bộ nhớ, thế nên ta vẫn có thể gọi được các phương thức khác của đối tượng

thread. Tuy nhiên khi gọi các phương thức của nó thì phương thức sẽ được thực thi ở trong thread gọi nó.

### 3.12.6 Cơ chế Synchronization and Locks trong lập trình đa luồng

- Bởi vì có nhiều thread cùng chạy đồng thời và có thể cùng làm một việc hoặc cùng truy xuất vào một nguồn tài nguyên bởi vậy mà chúng ta cần phải chú ý vấn đề các thread làm việc với nhau như thế nào.
- Giả sử như trong một phòng giao dịch của ngân hàng, có rất nhiều người cùng muốn được giao dịch, bạn cũng vậy. Và theo nguyên tắc ai tới trước thì được phục vụ trước. Tuy nhiên có tới 2 người cùng tới một lúc và thế là cuộc giằng co xảy ra, lúc này không ai trong hai người được phục vụ hết, thậm chí những người khác cũng không được phục vụ. Vậy giờ chúng ta cần có một giải pháp để giải quyết được điều này. Thông thường ở các điểm giao dịch ngân hàng đều đã trang bị một loại máy phát thẻ tự động. Trên thẻ có ghi số thứ tự phục vụ, ai bấm được thẻ trước thì được phục vụ trước. Trong lập trình Multi thread cũng vậy ta luôn luôn phải đồng bộ giữa các thread để giải quyết các tranh chấp một cách thỏa đáng bằng các kỹ thuật synchronization implemented bởi locks.
- Ta tìm hiểu ban đầu về kỹ thuật synchronization qua hình minh họa dưới đây.



- Khi Thread 1 muốn sử dụng Object 1 thì bắt đầu yêu cầu quyền sử dụng bằng cách Lock Object 1 lại. Thread khác vẫn làm công việc của mình. Trong thời gian Thread 1 chưa unlock Object 1 mà các Thread 2,3 cũng đồng thời muốn sử dụng Object 1 thì các Thread này sẽ phải trong trạng thái Blocked cho tới khi Thread 1 Unlock Object 1, và một trong hai Thread 2 hoặc 3 sẽ Lock được Object 1 và thực hiện tiếp công việc của mình.

### 3.12.6.1 Tình huống xung đột

- Ta xét cụ thể hơn về ví dụ code Java như sau để thấy việc không đồng bộ sẽ có kết quả ra sao:

```

10  * @author tanht
11  */
12  class Tracker {
13
14      public static void main(String[] args) {
15          Counter counterObj = new Counter();
16          CounterThread counterThreadObj1 = new CounterThread("Thread 1", counterObj);
17          CounterThread counterThreadObj2 = new CounterThread("Thread 2", counterObj);
18          CounterThread counterThreadObj3 = new CounterThread("Thread 3", counterObj);
19      }
20  }
21  public class Counter {
22      private int counter = 0;
23      public int getNextCounter() {
24          return counter++;
25      }
26  }
27
28  class CounterThread implements Runnable {
29
30      Thread myThread;
31      Counter counter;
32
33      CounterThread(String name, Counter couter) {
34          this.myThread = new Thread(this, name);
35          this.counter = couter;
36          this.myThread.start();
37      }
38
39      public void run() {
40          for (int i = 1; i <= 3; i++) {
41              System.out.println("Thread: "+myThread.getName()+" Count: "+counter.getNextCounter());
42          }
43      }
44  }

```

- Tại dòng 15 chúng ta khai báo duy nhất một biến đối tượng `counterObj` và khởi tạo đối tượng.
- Tại dòng 16,17,18 chúng ta khai báo 3 Thread và khởi chạy chúng ở constructor của chúng. Cả 3 Thread này đều sử dụng chung một đối tượng `counterObj`, đối tượng này được truyền vào từ Constructor.
- Tại dòng 24 là kết quả trả về tăng biến `counter` lên một đơn vị mỗi khi gọi method `getNextCounter`.
- Dòng 41 là lệnh gọi method `getNextCounter()` và hiển thị chúng ra màn hình. Và kết quả là:

```
Output - JavaCoreAdvance (run) X
run:
Thread: Thread 3 Count: 0
Thread: Thread 2 Count: 1
Thread: Thread 1 Count: 1
Thread: Thread 2 Count: 3
Thread: Thread 2 Count: 5
Thread: Thread 3 Count: 2
Thread: Thread 1 Count: 4
Thread: Thread 3 Count: 6
Thread: Thread 1 Count: 7
.
```

- o Quan sát kỹ kết quả thì có tới 2 giá trị 1 được lấy ra trên 2 Thread 1 và 2. Vậy là đã không đảm bảo tính liên tục của giá trị counter. Nguyên nhân là do Thread 1 và 2 cùng nhau tăng giá trị từ 0 lên 1 và lấy ra cùng 1 giá trị.

#### 3.12.6.2 Giải pháp giải quyết xung đột

- Để giải quyết xung đột từ ví dụ trên để đúng logic thì ta có phương án như sau:

  - o Cách 1 Class Lock: Thêm từ khóa synchronized vào khi khai báo method getNextCounter()

```
21  public class Counter {
22      private int counter = 0;
23      public synchronized int getNextCounter() {
24          return counter++;
25      }
26  }
```

➤ Với cách này thì toàn bộ method getNextCounter() sẽ được Lock lại và tại một thời điểm chỉ duy nhất 1 thread có thể tiến vào thực thi method getNextCounter(). Khi thực hiện xong method getNextCounter() thì method tự động được unlock và các thread khác có thể lại tiến tới, lock và thực thi method. vì vậy sẽ không thể xảy ra lỗi logic như ví dụ trên được nữa.

- o Cách 2: Object Lock: Thêm đoạn code dòng 25 như sau:

```
22  public class Counter {
23      private int counter = 0;
24      public int getNextCounter() {
25          synchronized (this) {
26              return counter++;
27          }
28      }
29  }
```

➤ Với cách này thì chỉ những đoạn code nào nằm trong {} của synchronized thì mới bị lock lại mỗi khi có 1 thread tiến vào thực hiện đoạn code này. Khi thực hiện xong đoạn code trong {} thì thread tự động Unlock đối tượng monitor cho thread khác tiến vào đoạn code trong {}, và vì vậy nên sẽ không thể xảy ra lỗi logic như ví dụ trên nữa. Đối tượng bên trong synchronized(Object) thực chất là một đối tượng dùng để monitor. Trường hợp này ta dùng luôn đối tượng counter bằng cách dùng từ khóa this. Hoặc chúng ta dùng một đối tượng khác giống như bên dưới như sau:

```

22  public class Counter {
23      private int counter = 0;
24      final Object monitorObj = new Object();
25      public int getNextCounter() {
26          synchronized (monitorObj) {
27              return counter++;
28          }
29      }
30  }

```

- Để tối ưu thì ta nên sử dụng cách 2 để lock một đoạn code, thường được gọi là đoạn mã găng để tăng hiệu suất của chương trình. Ta cần xác định chính xác đâu là đoạn găng của chương trình.
- Đoạn mã găng (critical section) là đoạn mã mà nhiều thread cùng ghi hoặc vừa đọc vừa ghi vào một hoặc nhiều đối tượng dùng chung ví dụ cùng tăng, giảm một giá trị của biến, cùng mở và ghi vào cùng một file, dùng chung cùng một socket...

### 3.12.7 Cơ chế Wait và Notify trong Java

- Trong quá trình hoạt động Thread thực thi các công việc(mã lệnh), theo logic nghiệp vụ thì đôi khi chúng cần phải chờ đợi một điều gì đó để có thể tiếp tục thực hiện công việc của mình. Ví dụ như quá trình một thread lấy dữ liệu ra từ một hàng đợi(Queue) và đem đi xử lý. Nếu dữ liệu trong Queue đã hết thì Thread cần phải chờ đợi. Thread sẽ chờ đợi cho tới khi có dữ liệu mới được đưa vào thì ngay sau đó nó chuyển sang trạng thái hoạt động để lấy dữ liệu ra và tiếp tục xử lý. Trong quá trình Thread chờ đợi thì nó ở trạng thái Waiting và không chiếm dụng tài nguyên CPU. Việc này làm cho chương trình trở nên chạy nhẹ nhàng hơn, tối ưu hơn.
- Sau đây ta xem xét một ví dụ về việc một Producer và Consumer hoạt động kết hợp với nhau và sử dụng cơ chế wait/notify để chờ đợi và báo hiệu cho nhau.



```

14  * @author tanth
15  */
16  public class WaitNotifyExample {
17
18      public static void main(String[] args) {
19          QueueData<Integer> queue = new QueueData();
20          ProducerThread p1 = new ProducerThread("Producer1", queue);
21          ConsumerThread c1 = new ConsumerThread("Consumer1", queue);
22          p1.start();
23          c1.start();
24      }
25  }
26
27  class QueueData<T> {
28      private final LinkedList<T> queue = new LinkedList<>();
29      public synchronized void put(T t) {
30          System.out.println("Producer is putting data into the Queue");
31          queue.addLast(t);
32          this.notify();
33      }
34
35      public synchronized T take() {
36          if (queue.isEmpty()) {
37              try {
38                  System.out.println("Consumer is waiting.....");
39                  this.wait();
40              } catch (InterruptedException ex) {
41                  System.out.println("Error: " + ex.getMessage());
42              }
43          }
44          System.out.println("Consumer is Polling data From Queue");
45          return queue.pollFirst();
46      }
47  }
  
```

```

48
49     class ProducerThread extends Thread {
50         QueueData<Integer> queue;
51     }
52     ProducerThread(String name, QueueData<Integer> queue) {
53         super(name);
54         this.queue = queue;
55     }
56     @Override
57     public void run() {
58         int i = 0;
59         while (i < Integer.MAX_VALUE) {
60             try {
61                 queue.put(i++);
62                 sleep(1000);
63             } catch (InterruptedException ex) {
64                 Logger.getLogger(ProducerThread.class.getName())
65                     .log(Level.SEVERE, null, ex);
66             }
67             System.out.println("Thread "+getName()+" put " + i + " to Queue");
68         }
69     }
70
71     class ConsumerThread extends Thread {
72
73         QueueData<Integer> queue;
74
75     }
76     ConsumerThread(String name, QueueData<Integer> queue) {
77         super(name);
78         this.queue = queue;
79     }
80
81     @Override
82     public void run() {
83         int i = 0;
84         while (true) {
85             i = queue.take();
86             System.out.println("Thread "+getName()+" take "+ i + " from Queue");
87         }
88     }

```

- o Một đối tượng QueueData được khởi tạo tại dòng 19
- o ProducerThread p1 và ConsumerThread c1 cùng được khởi tạo và truyền đối tượng QueueData vào Constructor, sau đó cả hai cùng được chạy tại dòng 22 và 23.
- o Nhiệm vụ của ProducerThread p1 là tạo ra các giá trị Integer để put vào Queuedata như dòng 60

- Nhiệm vụ của ConsumerThread c1 là chờ đợi khi QueueData có dữ liệu thì lấy ra và xuất ra màn hình như dùng 84.
- ProducerThread P1 có tốc độ put dữ liệu vào chậm hơn rất nhiều so với ConsumerThread c1 do nó sleep 1000ms sau mỗi lần put data(Dòng 61). Chính bởi vậy c1 sẽ phải đợi (tại dòng 39) để P1 put data vào thì mới lấy dữ liệu ra được ( tại dòng 45).
- Khi c1 đang chạy thì QueueData hết dữ liệu thì nó gọi method wait() để được chuyển sang trạng thái waiting và chờ đợi ProducerThread p1 put data vào QueueData như dòng 39
- Khi P1 put dữ liệu vào QueueData thì ngay sau đó nó thực hiện notify cho C1 như dòng 32. Lúc này C1 nhận được tín hiệu notify đang waiting liền chuyển sang trạng thái runnable để có thể tiếp tục lấy dữ liệu ra từ QueueData
- Dưới đây là kết quả chạy chương trình. Ta thấy Consumer thường xuyên phải ở trạng thái Waiting và sau đó lại tiếp tục Polling data ngay sau khi Producer put data vào Queue.

```
: Output - JavaCoreAdvance (run)
run:
Consumer is waiting.....
Producer is putting data into the Queue
Consumer is Polling data From Queue
Thread Consumer1 take 0 from Queue
Consumer is waiting.....
Thread Producer1 put 1 to Queue
Producer is putting data into the Queue
Consumer is Polling data From Queue
Thread Consumer1 take 1 from Queue
Consumer is waiting.....
Thread Producer1 put 2 to Queue
Producer is putting data into the Queue
Consumer is Polling data From Queue
Thread Consumer1 take 2 from Queue
Consumer is waiting.....
Thread Producer1 put 3 to Queue
Producer is putting data into the Queue
Consumer is Polling data From Queue
Thread Consumer1 take 3 from Queue
```

- Để một Thread chuyển sang trạng thái Waiting thì Thread sẽ triệu hồi method wait() của một đối tượng nào đó. Trong ví dụ trên thì đối tượng đó chính là Queue như tại dòng 39.

```

35  public synchronized T take() {
36      if (queue.isEmpty()) {
37          try {
38              System.out.println("Consumer is waiting.....");
39              this.wait();
40          } catch (InterruptedException ex) {
41              System.out.println("Error: " + ex.getMessage());
42          }
43      }
44      System.out.println("Consumer is Polling data From Queue");
45      return queue.pollFirst();
46  }

```

- Lời gọi method wait(), notify(), notifyAll() cần phải được đặt vào trong một khối lệnh đã được Synchronized như tại dòng 35.
- Để một Thread muốn chấm dứt trạng thái waiting thì đối tượng mà nó đã gọi wait() phải nhận được tín hiệu notify từ một lời gọi notify()/notifyAll() khác như tại dòng 32.

```

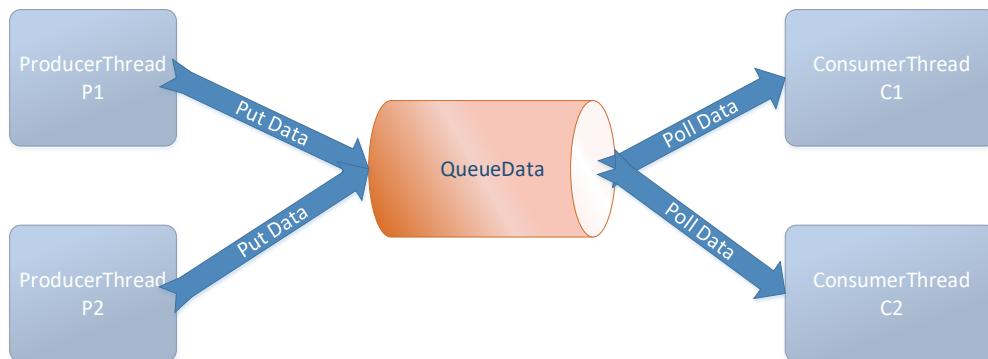
29  public synchronized void put(T t) {
30      System.out.println("Producer is putting data into the Queue");
31      queue.addLast(t);
32      this.notify();
33  }

```

- Khi một đối tượng nhận được notify thì một trong những Thread đang trong trạng thái waiting đối tượng đó sẽ nhận được tín hiệu notify và chuyển sang trạng thái runnable.

### 3.12.8 Cơ chế Wait và NotifyAll trong Java

- Cũng cùng ví dụ như trên nhưng chúng ta tăng số Thread lên như hình vẽ minh họa dưới đây.



```

15  * @author tanth
16  */
17  public class WaitNotifyAllExample {
18
19      public static void main(String[] args) {
20          QueueMaxData<Integer> queue = new QueueMaxData(1);
21          ProducerThread p1 = new ProducerThread("Producer1", queue);
22          ProducerThread p2 = new ProducerThread("Producer2", queue);
23          ConsumerThread c1 = new ConsumerThread("Consumer1", queue);
24          ConsumerThread c2 = new ConsumerThread("Consumer2", queue);
25          p1.start();
26          p2.start();
27          c1.start();
28          c2.start();
29      }
30  }
31
32  class QueueMaxData<T> {
33      private final LinkedList<T> queue = new LinkedList<>();
34      private int capacity = Integer.MAX_VALUE;
35      QueueMaxData(int capacity) {
36          this.capacity = capacity;
37      }
38      public synchronized void put(T t) {
39          while (queue.size() >= capacity) {
40              try {
41                  System.out.println("Producer is waiting");
42                  this.wait();
43              } catch (InterruptedException ex) {
44                  Logger.getLogger(QueueMaxData.class.getName())
45                      .log(Level.SEVERE, null, ex);
46              }
47          }
48          System.out.println("Producer is putting data into the Queue, size: "
49                          + queue.size());
50          queue.addLast(t);
51          this.notify();
52      }

```

```
53
54     public synchronized T take() {
55         T e = null;
56         while (e == null) {
57             if (queue.isEmpty()) {
58                 try {
59                     System.out.println("Consumer is waiting.....");
60                     this.wait();
61                 } catch (InterruptedException ex) {
62                     System.out.println("Error: " + ex.getMessage());
63                 }
64             }
65             System.out.println("Consumer is Polling data From Queue");
66             e = queue.pollFirst();
67             this.notifyAll();
68         }
69         return e;
70     }
71 }
72
73 class ProducerThread extends Thread {
74     QueueMaxData<Integer> queue;
75     ProducerThread(String name, QueueMaxData<Integer> queue) {
76         super(name);
77         this.queue = queue;
78     }
79
80     @Override
81     public void run() {
82         int i = 0;
83         while (i < Integer.MAX_VALUE) {
84             queue.put(i++);
85             System.out.println("Thread " + getName() + " put " + i + " to Queue");
86         }
87     }
88 }
```

```

89
90     class ConsumerThread extends Thread {
91         QueueMaxData<Integer> queue;
92         ConsumerThread(String name, QueueMaxData<Integer> queue) {
93             super(name);
94             this.queue = queue;
95         }
96         @Override
97         public void run() {
98             int i = 0;
99             while (true) {
100                 i = queue.take();
101                 System.out.println("Thread " + getName() + " take " + i + " from Queue");
102             }
103         }
104     }
105

```

- Xét ví dụ trên gần tương đồng với ví dụ về cơ chế notify trước đó.
  - Điểm khác biệt thứ nhất nằm ở chỗ có tới 2 ProducerThread p1, p2 và 2 ConsumerThread c1, c2.
  - Điểm khác biệt thứ 2 là ta thay thế QueueData bằng một QueueMaxData quy định số lượng phần tử tối đa được phép chứa trong Queue.
  - Điểm khác biệt thứ 3 là khi QueueMaxData đạt đến giá trị tối đa số lượng phần tử được phép trong queue thì p1,p2 phải đợi cho tới khi các consumerThread c1,c2 lấy ra các phần tử và notify cho p1,p2 thì lúc đó p1,p2 mới tiếp tục put thêm các phần tử vào QueueMaxData
  - Điểm khác biệt thứ 4 nằm ở chỗ dòng 67 là notifyAll() thay vì notify().
- Nếu dòng số 67 chúng ta dùng notify thay vì notifyAll() thì chương trình sẽ bị deadlock, tức là các Thread p1,p2, c1,c2 đều bị dừng và không thể chạy tiếp được mặc dù chương trình chưa kết thúc. Điều kì cục này được giải thích như sau:
  - Khi QueueMaxData đạt tới ngưỡng max số phần tử thì p1, p2 sẽ rơi vào trạng thái waiting.
  - Sau đó c1 và c2 lấy ra các phần tử trong QueueMaxData và thực hiện gọi method notify(). Lúc này cả 4 thread p1,p2,c1,c2 đều có cơ hội nhận về tín hiệu notify là như nhau vì vậy sẽ có những tình huống chính c1 và c2 nhận được notify của chính nó hoặc của nhau. Nếu xảy ra như vậy thì p1 và p2 sẽ mãi mãi không được notify nữa và đồng nghĩa với mãi mãi ở trong trạng thái waiting. Dẫn đến c1, c2 cũng sẽ mãi mãi ở trạng thái waiting. Vì vậy các thread này đã cùng lúc chuyển về trạng thái waiting. Và đây là một tình huống Missing Sinal.
  - Để khắc phục tình trạng này thì tại dòng 67 thay vì dùng notify() thì chúng ta dùng notifyAll() thì tình huống deadlock sẽ không xảy ra nữa do cả 4 thread p1,p2,c1,c2 đều nhận được tín hiệu notify và cùng chuyển trạng thái để tiếp tục thực hiện công việc.

- Do cùng lúc cả 4 thread đều cùng nhận được notify và cùng chuyển trạng thái làm việc vì vậy để tránh lỗi logic thì chúng ta cần chú ý điều này và thực hiện kiểm tra điều kiện logic để tiếp tục đưa các thread về đúng trạng thái cần thiết.
- Khi một đối tượng nhận được notify thì tất cả những Thread đang trong trạng thái waiting đối với đối tượng đó sẽ đều nhận được tín hiệu notify và chuyển sang trạng thái runnable. Điều này khác với notify chỉ có 1 thread nhận được notify. Chính điều này làm nên sự khác biệt trong cơ chế notifyAll mà chúng ta cần phải chú ý.

### 3.12.9 Độ ưu tiên của Thread.

- Giả sử ta có một công việc cần phải thực hiện định kì và cần đúng thời điểm dạng như một scheduler. Để làm được việc này ta tạo ra một thread và thực hiện định kì công việc sau mỗi khi sleep(x). Vấn đề đặt ra là sau mỗi khi sleep x time thì thread sẽ được quay trở lại trạng thái runnable mà chưa phải là running vì vậy mà việc thực hiện công việc chuẩn xác thời gian sau x time không được đảm bảo.
- Để đảm bảo cho việc scheduler thực hiện đúng giờ thì ta cần phải nâng độ ưu tiên của thread lên để rút ngắn độ trễ khi chuyển từ trạng thái runnable sang running. Method setPriority giúp chúng ta làm được điều này.

**public final void setPriority(int newPriority)**

Trong đó từ 1 đến 10 là các mức độ ưu tiên của một thread. Giá trị này càng lớn thì Thread càng được ưu tiên thực hiện trước các Thread khác.

```

    /**
     * The minimum priority that a thread can have.
     */
    public final static int MIN_PRIORITY = 1;

    /**
     * The default priority that is assigned to a thread.
     */
    public final static int NORM_PRIORITY = 5;

    /**
     * The maximum priority that a thread can have.
     */
    public final static int MAX_PRIORITY = 10;

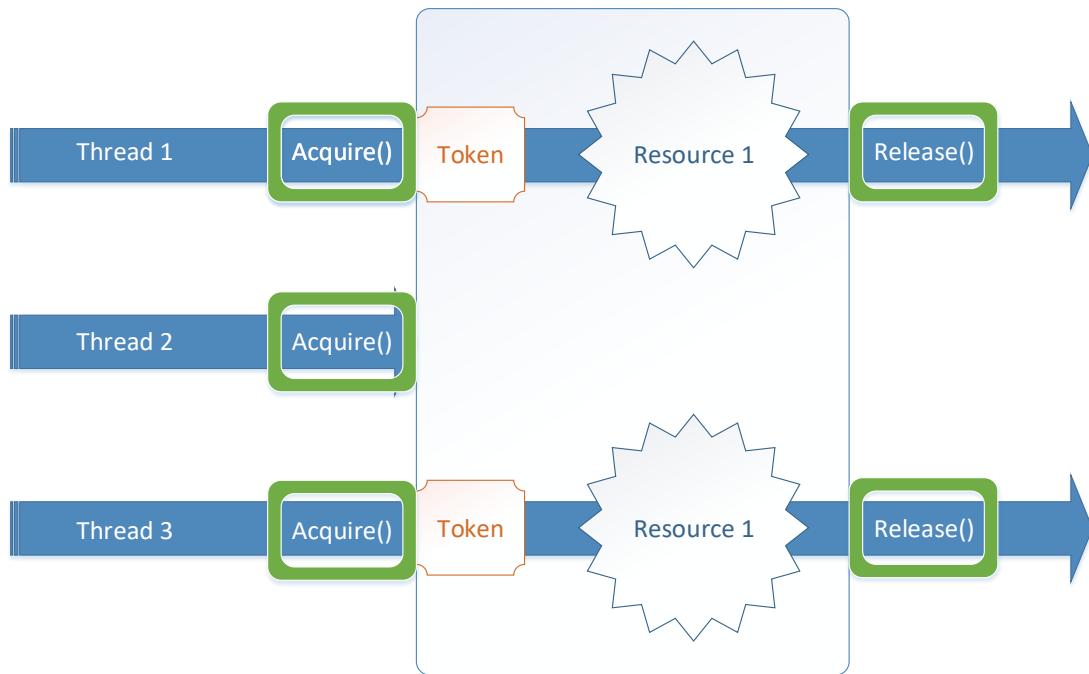
```

### 3.12.10 Cơ chế đồng bộ Semaphore.

- Khi một nhóm gồm 10 người chúng ta đi chơi mà chỉ mang có 5 chai nước suối. Điều gì sẽ xảy ra khi 10 người cùng dừng lại và cùng lấy nước để uống. Tất nhiên là số chai nước suối là không đủ cho 10 người và 5 người trong số chúng ta sẽ phải chờ cho tới khi lần lượt 5 người trước uống xong thì 5 người còn lại mới

có thể tiếp cận các chai nước và uống chúng. Vậy số chai nước là nguồn tài nguyên hữu hạn trong khi nhu cầu sử dụng nguồn tài nguyên này cùng một lúc lại nhiều hơn. Chính vì vậy trong đời sống cũng như trong chương trình chúng ta đều phải có cơ chế để phân xử điều này.

- Đối với chương trình java, khi chúng ta có số lượng tài nguyên biến, đối tượng... hữu hạn và chúng ta cần giới hạn số lượng các thread cùng sử dụng số tài nguyên này thì chúng ta sử dụng đối tượng Semaphore để thực hiện đồng bộ giữa các thread điểu này giúp tránh tranh chấp tài nguyên. Hình dưới đây là minh họa cho câu chuyện về cơ chế đồng bộ Semaphore.



- Chú ta đi xét ví dụ sau: Chương trình cho phép tạo ra 3 thread để thực hiện download, tuy nhiên lại chỉ có thể tạo được tối đa 2 connection để download, vì vậy phải có 1 thread sẽ phải chờ 2 thread download xong thì mới có thể download được và cứ phải chờ nhau như vậy. Dưới đây là code minh họa.

```

17  * @author tanth
18  */
19  public class SemaphoreExample {
20      static DownloadManager manager = new DownloadManager(2);
21      public static void main(String[] args) {
22          DownloadThread downloadThread1 = new DownloadThread("downloadThread1");
23          DownloadThread downloadThread2 = new DownloadThread("downloadThread2");
24          DownloadThread downloadThread3 = new DownloadThread("downloadThread3");
25          downloadThread1.start();
26          downloadThread2.start();
27          downloadThread3.start();
28      }
29  }

```

```

30
31     class DownloadManager {
32         int maxConnection = 0;
33         int count = 0;
34         Semaphore semaphore;
35         DownloadManager(int maxConnection) {
36             this.maxConnection = maxConnection;
37             semaphore = new Semaphore(maxConnection);
38         }
39         public void Download(String url) throws InterruptedException {
40             semaphore.acquire();
41             System.out.println("Downloading...");
42             Connection connection = new Connection();
43             connection.download(url);
44             semaphore.release();
45         }
46     }
47
48     class DownloadThread extends Thread {
49         DownloadThread(String name) {
50             super(name);
51         }
52         @Override
53         public void run() {
54             while(true){
55                 try {
56                     System.out.println("Thread " + getName() + " Require download");
57                     manager.Download("http://activestudy.online/javabook.pdf");
58                 } catch (InterruptedException ex) {
59                     Logger.getLogger(DownloadThread.class.getName())
60                         .log(Level.SEVERE, null, ex);
61                 }
62             }
63         }
64     }
65
66     class Connection {
67         File download(String url) throws InterruptedException{
68             //Example code here
69             sleep(5000);
70             return null;
71         }
72     }
73

```

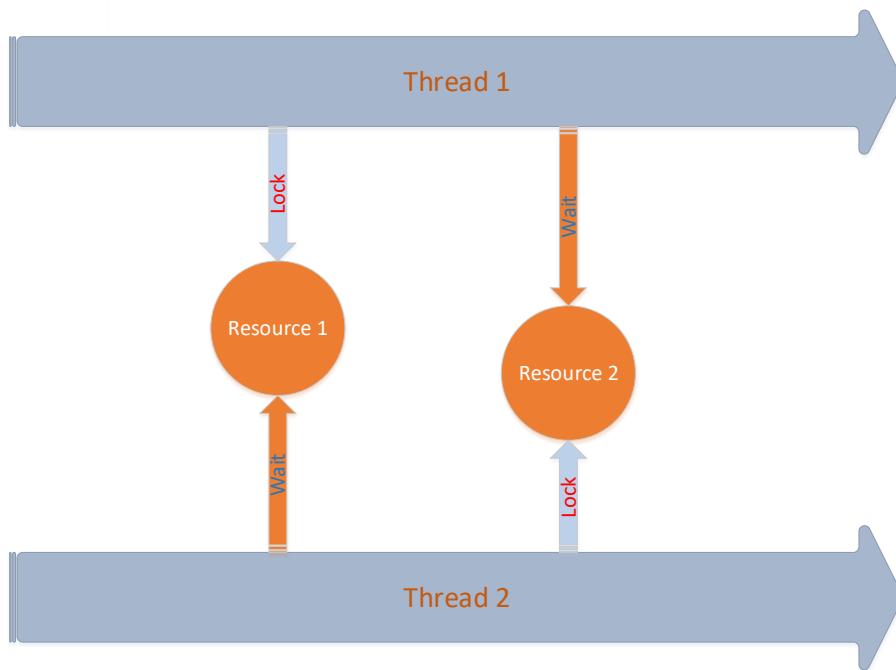
- Dòng 20 khởi tạo đối tượng downloadManager với tối đa 2 connection.
- Dòng 22 tới dòng 27 là quá trình khởi tạo 3 thread DownloadThread để tiến hành download.
- Dòng 37 khởi tạo đối tượng Semaphore với permit = maxConnection. Có nghĩa là Semaphore này chỉ cho phép tối đa 2 thread tiến vào sử dụng tài nguyên.

- o Dòng 40 Semaphore thực hiện acquire(). Tại đây Semaphore kiểm tra nếu chưa vượt quá số lượng thread đang trong vùng găng thì Semaphore sẽ cấp thẻ để thread tiến vào. Nếu đã vượt quá số lượng thread đang trong vùng găng thì thread sẽ phải chờ cho tới khi có một thread release tài nguyên vùng găng mới có thể tiến vào vùng găng để thực hiện và sử dụng tài nguyên vùng găng.
- o Dòng 44 thực hiện release tài nguyên vùng găng sau khi đã thực hiện xong. Nếu ta quên thao tác này thì sẽ không có thêm thread nào tiến vào vùng găng sử dụng tài nguyên được nữa và sẽ dẫn đến deadlock.
- o Kết quả chạy chương trình cho thấy, có 3 thread cùng require download nhưng chỉ có 2 thread tiến vào download còn 1 thread phải đợi cho tới khi download xong mới có thể tiến vào download.

```
: Output - JavaCoreAdvance (run)
run:
Thread downloadThread2 Require download
Thread downloadThread1 Require download
Thread downloadThread3 Require download
Downloading...
Downloading...
Thread downloadThread1 Require download
Downloading...
Downloading...
Thread downloadThread2 Require download
Downloading...
Thread downloadThread3 Require download
Thread downloadThread1 Require download
Downloading...
Thread downloadThread3 Require download
```

### 3.12.11 Deadlock và các kỹ thuật tránh deadlock

- Deadlock là hiện tượng các thread cùng sử dụng chung và chờ đợi tài nguyên của nhau theo một vòng tròn dẫn tới tất cả đều không thể tiếp tục được thực thi.



- Chúng ta đi xét ví dụ về deadlock đợi tài nguyên theo vòng tròn như sau:

```

14  * @author tanth
15  */
16 public class DeadLockExample {
17
18     public static void main(String[] args) {
19         Object monitorObj1 = new Object();
20         Object monitorObj2 = new Object();
21         myThread p1 = new myThread("Thread1", monitorObj1, monitorObj2);
22         myThread p2 = new myThread("Thread2", monitorObj2, monitorObj1);
23         p1.start();
24         p2.start();
25     }
26 }
```

```

27
28     class myThread extends Thread {
29
30         Object monitorObj1;
31         Object monitorObj2;
32
33     public myThread(String name, Object monitorObj1, Object monitorObj2) {
34         super(name);
35         this.monitorObj1 = monitorObj1;
36         this.monitorObj2 = monitorObj2;
37     }
38
39     @Override
40     public void run() {
41         while (true) {
42             System.out.println(getName() + "wait for a lock on monitorObj1");
43             synchronized (monitorObj1) {
44                 System.out.println(getName() + "locked on monitorObj1");
45                 System.out.println(getName() + "wait for a lock on monitorObj2");
46                 synchronized (monitorObj2) {
47                     System.out.println(getName() + "locked on monitorObj2");
48                 }
49                 try {
50                     Thread.sleep(500);
51                 } catch (InterruptedException ex) {
52                     Logger.getLogger(myThread.class.getName())
53                         .log(Level.SEVERE, null, ex);
54                 }
55             }
56         }
57     }
58 }
```

- o Tại dòng 21 và 22 chúng ta khởi tạo 2 thread và truyền vào lần lượt 2 đối tượng dùng để làm LockObj, tuy nhiên điều chú ý là chúng đã được đổi chéo thứ tự truyền vào Thread ở hai dòng 21 và 22, và chúng ta chờ xem điều gì sẽ xảy ra.
- o Khi 2 thread cùng khởi chạy thì chúng sẽ lần lượt đợi LockObj của nhau. Trong khi thread P1 Lock monitorObject và wait monitorObj2 thì thread P2 lại làm điều ngược lại là Lock monitorObj2 và wait monitorObj1. Và vì vậy đây là tình huống lock phỗ biến và đơn giản nhất. Và bên dưới đây là kết quả sau khi chạy chương trình, hai thread p1 và p2 đã bị dừng lại và không thể chạy tiếp tục được nữa.

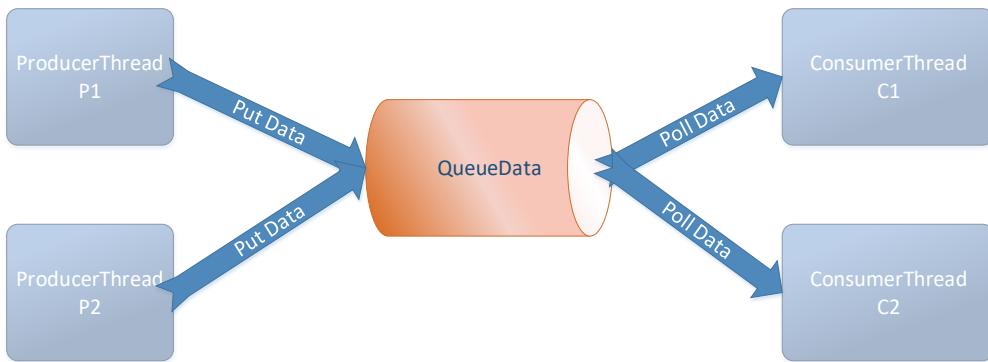
```
: Output - JavaCoreAdvance (run)
run:
Thread1wait for a lock on monitorObj1
Thread2wait for a lock on monitorObj1
Thread1locked on monitorObj1
Thread2locked on monitorObj1
Thread2wait for a lock on monitorObj2
Thread1wait for a lock on monitorObj2
```

- Trong các chương trình do số lượng thread nhiều hơn rất nhiều, và các ObjectLock lại đa dạng vì vậy để phát hiện ra các deadlock ko hề dễ dàng và nhiều lúc cần phải cho những chuyên gia giàu kinh nghiệm thì mới có thể phát hiện được deadlock.
- Khi đã hiểu được deadlock thì chúng ta cần phải tránh được các tình huống deadlock. Một quy tắc khi sử dụng các ObjectLock để tránh deadlock đó là chương trình phải viết để Lock các ObjectLock một cách có thứ tự, đó là Lock ObjecLock1 xong mới lock đến ObjectLoc2 và thứ tự mọi nơi luôn luôn là như vậy hoặc ngược lại Lock ObjecLock2 xong mới lock đến ObjectLoc1. Nếu làm như vậy thì chúng ta không bao giờ bị deadlock nữa cả. Vì vậy ví dụ ở trên nếu sửa như sau thì sẽ ko bị deadlock nữa:

```
14  * @author tanth
15  */
16  public class DeadLockExample {
17
18  public static void main(String[] args) {
19      Object monitorObj1 = new Object();
20      Object monitorObj2 = new Object();
21      myThread p1 = new myThread("Thread1", monitorObj1, monitorObj2);
22      myThread p2 = new myThread("Thread2", monitorObj1, monitorObj2);
23      p1.start();
24      p2.start();
25  }
26 }
```

### 3.12.12 missing signal

- Đây là hiện tượng cũng làm cho các thread bị dừng lại do hiện tượng wait một object mãi mãi mà không nhận được tín hiệu notify do tín hiệu notify đã được nhận bởi một thread khác do notify nhầm.
- Chúng ta đi xem xét lại ví dụ đã nghiên cứu ở bài học về notifyAll, hình minh họa như bên dưới đây.



```

14     * @author tanth
15     */
16
17     public class MissingSinalExample {
18
19         public static void main(String[] args) {
20             QueueMaxData<Integer> queue = new QueueMaxData(1);
21             ProducerThread p1 = new ProducerThread("Producer1", queue);
22             ProducerThread p2 = new ProducerThread("Producer2", queue);
23             ConsumerThread c1 = new ConsumerThread("Consumer1", queue);
24             ConsumerThread c2 = new ConsumerThread("Consumer2", queue);
25             p1.start();
26             p2.start();
27             c1.start();
28             c2.start();
29         }
30
31
32         class QueueMaxData<T> {
33             private final LinkedList<T> queue = new LinkedList<>();
34             private int capacity = Integer.MAX_VALUE;
35             QueueMaxData(int capacity) {
36                 this.capacity = capacity;
37             }
38             public synchronized void put(T t) {
39                 while (queue.size() >= capacity) {
40                     try {
41                         System.out.println("Producer is waiting");
42                         this.wait();
43                     } catch (InterruptedException ex) {
44                         Logger.getLogger(QueueMaxData.class.getName())
45                             .log(Level.SEVERE, null, ex);
46                     }
47                 }
48                 System.out.println("Producer is putting data into the Queue, size: "
49                               + queue.size());
50                 queue.addLast(t);
51                 this.notify();
52             }
53         }
  
```

```
53
54     public synchronized T take() {
55         T e = null;
56         while (e == null) {
57             if (queue.isEmpty()) {
58                 try {
59                     System.out.println("Consumer is waiting.....");
60                     this.wait();
61                 } catch (InterruptedException ex) {
62                     System.out.println("Error: " + ex.getMessage());
63                 }
64             }
65             System.out.println("Consumer is Polling data From Queue");
66             e = queue.pollFirst();
67             this.notify();
68         }
69         return e;
70     }
71 }
72
73 class ProducerThread extends Thread {
74     QueueMaxData<Integer> queue;
75     ProducerThread(String name, QueueMaxData<Integer> queue) {
76         super(name);
77         this.queue = queue;
78     }
79
80     @Override
81     public void run() {
82         int i = 0;
83         while (i < Integer.MAX_VALUE) {
84             queue.put(i++);
85             System.out.println("Thread "+getName() + " put " + i + " to Queue");
86         }
87     }
88 }
```

```

89
90     class ConsumerThread extends Thread {
91         QueueMaxData<Integer> queue;
92         ConsumerThread(String name, QueueMaxData<Integer> queue) {
93             super(name);
94             this.queue = queue;
95         }
96         @Override
97         public void run() {
98             int i = 0;
99             while (true) {
100                 i = queue.take();
101                 System.out.println("Thread "+getName()+" take "+ i +" from Queue");
102             }
103         }
104     }

```

- o Tại dòng số 67 chúng ta đã sai lầm khi dùng notify thay vì notifyAll. Tình huống này đã được chỉ ra tại phần nghiên cứu về “Cơ chế Wait và NotifyAll trong Java”
- Kết quả chương trình đang chạy và các Thread bị dừng lại và không thể tiếp tục xử lý.

: Output - JavaCoreAdvance (run)

```

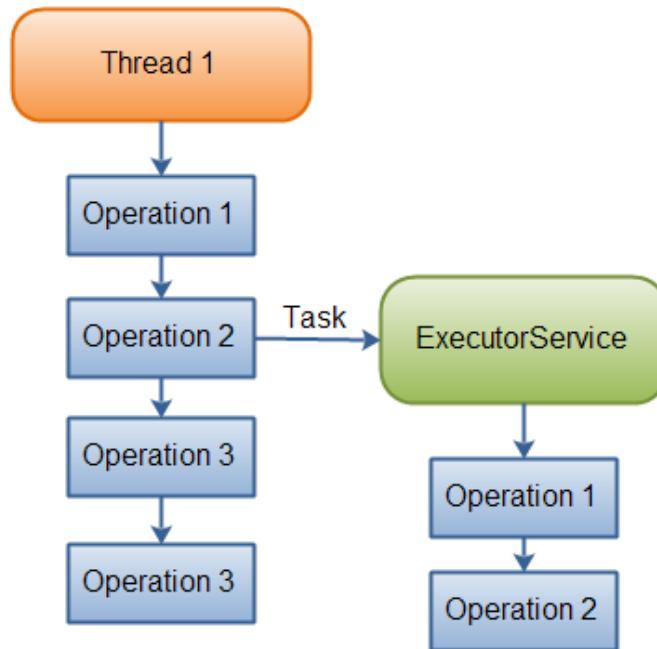
Thread Producer1 put 977 to Queue
Thread Producer1 put 978 to Queue
Thread Producer1 put 979 to Queue
Thread Producer1 put 980 to Queue
Thread Producer1 put 981 to Queue
Thread Producer1 put 982 to Queue
Thread Producer1 put 983 to Queue
Thread Producer1 put 984 to Queue
Thread Producer1 put 985 to Queue
Thread Producer1 put 986 to Queue
Thread Producer1 put 987 to Queue
Consumer wait.....
Thread Consumer2 take 976 from Queue
Thread Producer1 put 988 to Queue
wait to put

```

- Vậy để tránh tình huống missing signal này chúng ta sử dụng notifyAll() thay vì notify như ở tại dòng 67.

### 3.12.13 Thread pool trong Java

- Thay vì phải liên tục tạo mới và hủy đi thread thì chúng ta sử dụng thread pool để không bị tốn tài nguyên. Một khác chúng ta cũng giới hạn được số lượng Thread cần thiết để không tạo ra quá nhiều thread trong hệ thống mà ko kiểm soát được.



- Có một vài cách khác nhau để giao nhiệm vụ tới một ExecutorService:
  - o execute(Runnable)
  - o submit(Runnable)
  - o submit(Callable)
  - o invokeAny(...)
  - o invokeAll(...)

#### 3.12.13.1 Sử dụng phương thức execute (runnable) của ExecutorService

- Đây là phương thức được sử dụng khi ta cần giao nhiệm vụ cho ExecutorService thông qua một runnable mà không cần nhận về kết quả trả về.
- Ta đi xét ví dụ về execute() như sau:

```

15  * @author tanth
16  */
17  public class ThreadPoolExample {
18
19      public static void main(String[] args) {
20
21          ExecutorService executorService = Executors.newFixedThreadPool(3);
22          int i =0;
23          while(i++<100) {
24              executorService.execute(new myRunnable("Thread" + i));
25          }
26          executorService.shutdown();
27      }
28  }
29
30  class myRunnable implements Runnable{
31      String name;
32      myRunnable(String name){
33          this.name = name;
34      }
35      @Override
36      public void run() {
37          System.out.println("Thread "+ name+" is running task");
38          try {
39              Thread.sleep(5000);
40          } catch (InterruptedException ex) {
41              Logger.getLogger(myRunnable.class.getName())
42                  .log(Level.SEVERE, null, ex);
43          }
44      }
45  }
46

```

- Kết quả thực hiện chương trình như sau:

Output - JavaCoreAdvance (run)

```

run:
Thread Thread1 is running task
Thread Thread2 is running task
Thread Thread3 is running task
Thread Thread4 is running task
Thread Thread5 is running task
Thread Thread6 is running task
Thread Thread7 is running task
Thread Thread8 is running task
Thread Thread9 is running task
Thread Thread10 is running task
Thread Thread11 is running task

```

### 3.12.13.2 Sử dụng phương thức submit(Callable) của ExecutorService

- Đây là phương thức được sử dụng khi ta cần giao nhiệm vụ cho ExecutorService thông qua một Callable và cần nhận về kết quả sau khi thực thi.
- Kết quả sau khi thực thi của mỗi thread sau khi gọi submit(Callable) là một đối tượng kiểu Future<T>.
- Kiểu Future cho phép chúng ta lấy về kết quả thực hiện thông qua method get()
- Ta đi xét ví dụ về submit () để tính tổng các giai thừa của các số từ 1 tới n như sau:

```

20  * @author tanth
21  */
22  public class ThreadPoolSubmitExample {
23      public static void main(String[] args) {
24          int n = 20;
25          ExecutorService myPool = Executors.newFixedThreadPool(10);
26          ArrayList<Future> futures = new ArrayList<>();
27          for (int i = 1; i <= n; i++) {
28              Future<Long> future = myPool.submit(new myCallable(i));
29              futures.add(future);
30          }
31          myPool.shutdown();
32          Long sum = 0L;
33          try {
34              while (myPool.awaitTermination(1, TimeUnit.SECONDS) == false);
35              for (Future<Long> futureE : futures) {
36                  sum += futureE.get();
37              }
38          } catch (InterruptedException | ExecutionException ex) {}
39          System.out.println("Sum of Factorial is: " + sum);
40      }
41  }

```

```

42
43     class myCallable implements Callable<Long> {
44
45         int n;
46
47     public myCallable(int n) {
48         this.n = n;
49     }
50
51     @Override
52     public Long call() throws Exception {
53         return calculateFactorial(n);
54     }
55
56     public Long calculateFactorial(int n) {
57         return (n == 1) ? 1 : n * calculateFactorial(n - 1);
58     }
59 }
60

```

- Kết quả thực hiện chương trình như sau:

: Output - JavaCoreAdvance (run)

```

run:
Sum of Factorial is: 2561327494111820313
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Phương pháp dùng submit có thể được ứng dụng trong các trường hợp tính toán phân tán nhiều thread, nhiều hệ thống khác nhau cùng thực hiện 1 công việc và kết quả sẽ được tập hợp sau khi có kết quả.
- Tương tự như submit(Callable), phương thức **submit(Runnable)** cũng đưa vào 1 Runnable và nó trả về một đối tượng **Future**. Đối tượng Future có thể được sử dụng để kiểm tra nếu Runnable đã hoàn tất việc thực thi nhưng không thể trả về kết quả xử lý bằng method get().

### 3.12.13.3 Sử dụng phương thức invokeAny của ExecutorService

- Method **invokeAny()** trả về một **Future**, nhưng trả về kết quả chỉ là của **một trong những đối tượng Callable**. Nó không đảm bảo về kết quả bạn sẽ nhận được từ một callable nào, chỉ cần một trong số chúng hoàn thành. Tức là ko cần tất cả các Thread hoàn thành, chỉ cần 1 task hoàn thành phương thức get() sẽ nhận được kết quả.
- Nếu 1 trong số task hoàn thành hoặc ném ra 1 ngoại lệ, phần còn lại của Callable sẽ được hủy bỏ (cancelled).
- Phần này có thể được ứng dụng trong các trường hợp tính toán phân tán tìm ra kết quả nhanh nhất hoặc giải thuật tối ưu nhất.
- Xét lại ví dụ đã đề cập ở phần submit(Callable). Bây giờ chúng ta sẽ viết lại như sau:

```

19  * @author tanth
20  */
21  public class ThreadPoolInvokeAnyExample {
22      public static void main(String[] args) {
23          int n = 20;
24          try {
25              ExecutorService myPool = Executors.newFixedThreadPool(10);
26              List<Callable<Long>> callables = new ArrayList<>();
27              for (int i = 1; i <= n; i++) {
28                  callables.add(new myCallAble(i));
29              }
30              Long result = myPool.invokeAny(callables);
31              myPool.shutdown();
32              System.out.println("Factorial one of callables is: " + result);
33          } catch (InterruptedException | ExecutionException ex) {/**/}
34      }
35  }

```

- Kết quả thực thi chương trình sẽ nhận được kết quả tính giai thừa của một trong những số nằm trong khoảng từ 1 tới N và chúng ta không thể biết được nó là của thread nào thực thi và giai thừa của số nào.

Output - JavaCoreAdvance (run)

```

run:
Factorial one of callables is: 3628800

```

### 3.12.13.4 Sử dụng phương thức invokeAll của ExecutorService

- Phương thức **invokeAll()** gọi tất cả đối tượng Callable trong tập hợp. Phương thức này trả về 1 danh sách các đối tượng **Future<T>** mà được trả về từ việc thực thi các Callables. Kết quả sẽ được trả về chỉ khi tất cả các task trong các thread đều được thực hiện xong hoặc hết thời gian timeOut nếu chúng ta thiết đặt tham số này.
- Xét lại ví dụ đã đề cập ở phần submit(Callable). Jetzt giờ chúng ta sẽ viết lại như sau:

```

21  L  * @author tanth */
22  public class ThreadPoolInvokeAllExample {
23    public static void main(String[] args) {
24      int n = 20;
25      try {
26        ExecutorService myPool = Executors.newFixedThreadPool(10);
27        List<Callable<Long>> callables = new ArrayList<>();
28        List<Future<Long>> futures;
29        for (int i = 1; i <= n; i++) {
30          callables.add(new myCallAble(i));
31        }
32        futures = myPool.invokeAll(callables);
33        myPool.shutdown();
34        Long sum = 0L;
35        while (myPool.awaitTermination(1, TimeUnit.SECONDS) == false);
36        for (Future<Long> futureE : futures) {
37          sum += futureE.get();
38        }
39        System.out.println("Sum of Factorial is: " + sum);
40      } catch (InterruptedException | ExecutionException ex) {/**/}
41    }
42  }

```

- Kết quả thực thi chương trình như sau:

: Output - JavaCoreAdvance (run)

```

run:
Sum of Factorial is: 2561327494111820313

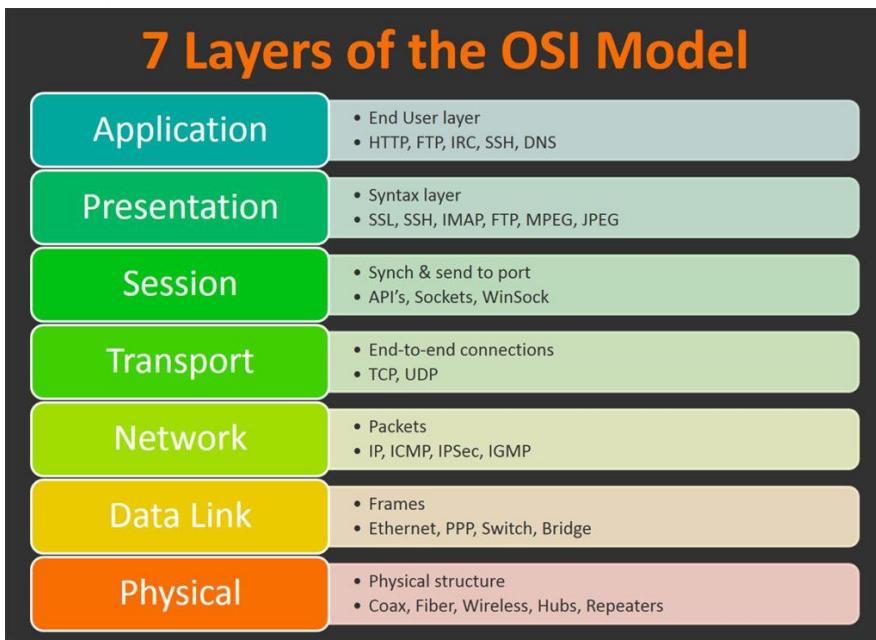
```

### 3.13 Lập trình mạng (Networking) trong Java:

#### 3.13.1 Các khái niệm mạng (networking) cơ bản

- Trước khi đi tìm hiểu và lập trình mạng chúng ta nên tìm hiểu về các khái niệm để hiểu về cơ chế kết nối và truyền nhận dữ liệu, các giao thức mạng phổ biến và hay dùng. Và sau đây chúng ta đi tìm hiểu về mô hình mạng nổi tiếng và phổ biến đó là mô hình 7 lớp OSI.

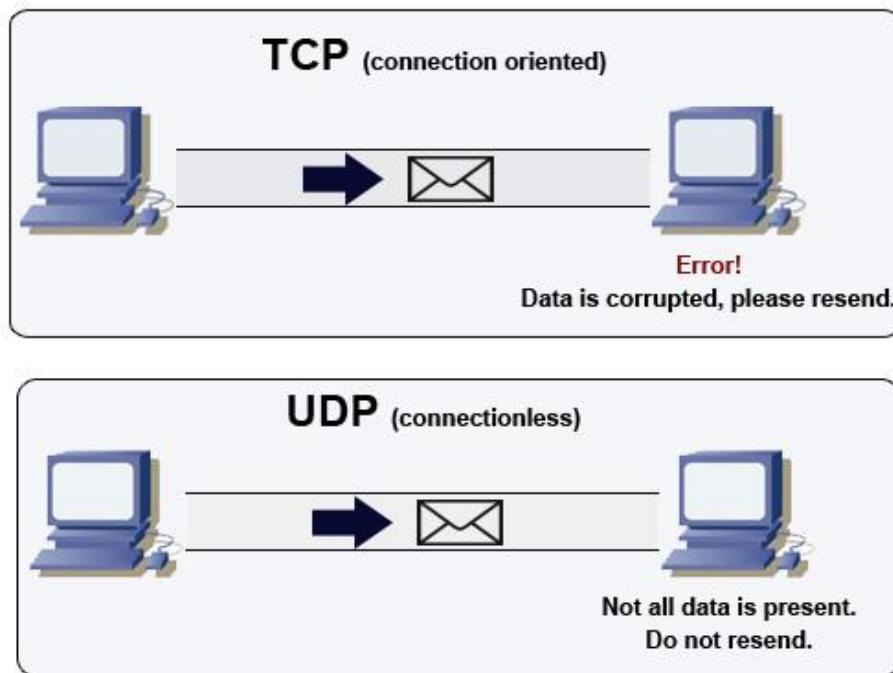
##### 3.13.1.1 Mô hình mạng 7 lớp OSI



- Bất kì một kết nối mạng nào cũng đang đều được xây dựng dựa vào việc tham chiếu theo mô hình 7 lớp OSI.(Hoặc mô hình tương đồng là TCP/IP)
  - o Tầng 1: Tầng vật lý (Physical) làm các nhiệm vụ kết nối vật lý giữa các thiết bị mạng ví dụ như Card mạng, dây mạng... những cái gì là cầm nắm được để đảm bảo tín hiệu điện được thông suốt thì đều nằm ở tầng này.
  - o Tầng 2: Tầng liên kết dữ liệu (Data Link): Là tầng chịu trách nhiệm truyền dữ liệu giữa hai nút mạng (liền nhau) dựa trên địa chỉ MAC và cung cấp các giao thức sửa chữa lỗi truyền dữ liệu của tầng vật lý. Dữ liệu truyền nhận ở tầng 2 có dạng frame. Các giao thức được dùng ở tầng này bao gồm PPP, HDLC... Các thiết bị có thể làm việc ở tầng này đó là: Card Ethernet, Switch, Bridge.
  - o Tầng 3: Tầng mạng (Network): Tầng này phụ trách truyền dữ liệu dưới dạng các gói dữ liệu (package) và phụ trách chuyển mạch, điều khiển lưu lượng và định tuyến các gói dữ liệu dựa vào địa chỉ IP.
  - o Tầng 4: Tầng giao vận (Transport): Tầng này có nhiệm vụ ghép, tách dữ liệu của các dịch vụ khác nhau để truyền dẫn trên cùng một đường truyền. Tầng này cung cấp hai giao thức truyền nhận dữ liệu bao gồm:
    - TCP : (Transmission Control Protocol)
    - UDP : (User Datagram Protocol)

No	Đặc điểm	TCP	UDP
1.	Kích thước header bản tin lớn (20 byte)	Lớn (20 Byte)	Nhỏ (8 byte)
2.	Tạo kết nối trước khi truyền dữ liệu	Yes	No
3.	Có phát hiện lỗi sai lệch hoặc mất mát dữ liệu	Yes	No
4.	Có truyền lại những dữ liệu lỗi	Yes	No
5.	Đảm bảo tính toàn vẹn và tin cậy của dữ liệu	Yes	No
6.	Tốc độ truyền tải dữ liệu	thấp	cao
7.	Không đảm bảo thời gian thực	No	Yes
8.	Áp dụng trong các trường hợp ứng dụng như	chat, báo hiệu...	VoiP, Video call...
9.	Môi trường phù hợp	WAN	LAN

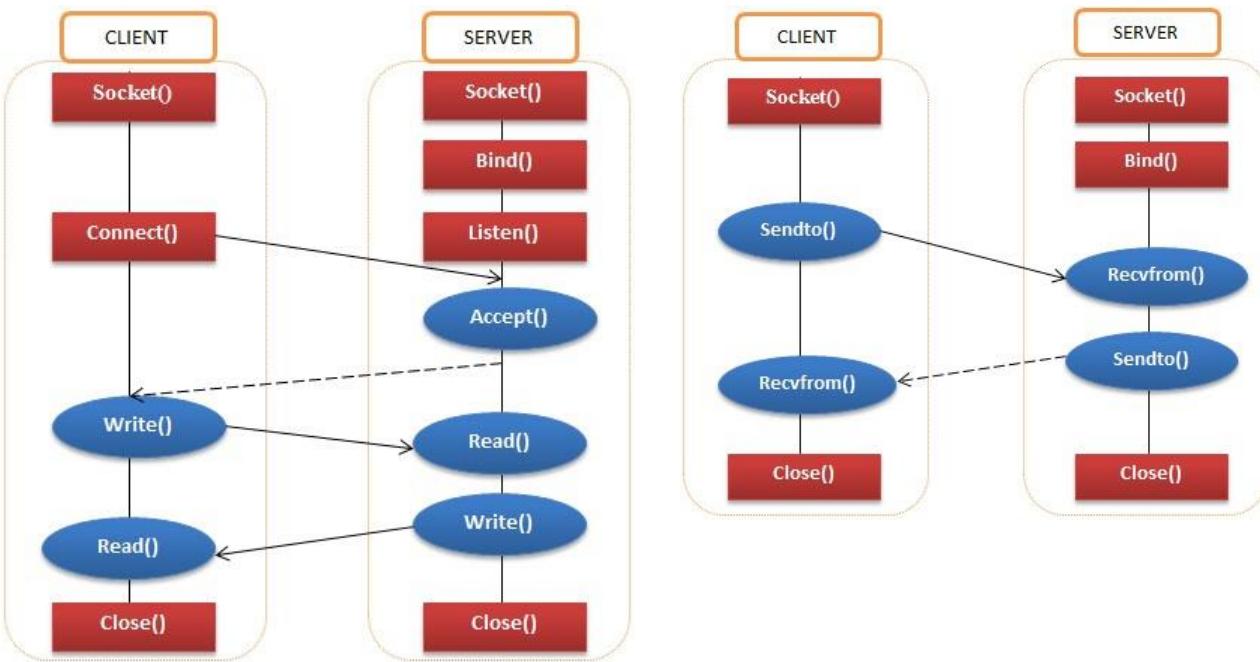
o



- o Tầng 5: Tầng phiên (Session): Đảm bảo dữ liệu từ các ứng dụng(Tiến trình) khác nhau có thể về đúng với với ứng dụng đó thông qua các port.
- o Tầng 6: Tầng trình diễn (Presentation)
- o Tầng 7: Tầng ứng dụng.

### 3.13.1.2 *Socket là gì*

- Socket là một cổng logic mà một ứng dụng tạo ra và sử dụng để kết nối với một ứng dụng khác chạy trên môi trường mạng. Một ứng dụng có thể sử dụng nhiều Socket cùng một lúc, nhờ đó nhiều ứng dụng có thể cùng sử dụng network cùng một lúc. Chúng ta có 2 loại Socket như sau:
  - o Stream Socket: Dựa trên giao thức TCP( Tranmission Control Protocol) việc truyền dữ liệu chỉ thực hiện giữa 2 quá trình đã thiết lập kết nối (Connection Oriented). Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn.
  - o Datagram Socket: Dựa trên giao thức UDP( User Datagram Protocol) việc truyền dữ liệu không yêu cầu có sự thiết lập kết nối giữa 2 quá trình (Connectionless). Ngược lại với giao thức TCP thì dữ liệu được truyền theo giao thức UDP không được tin cậy, có thể không đúng trình tự và lặp lại. Tuy nhiên vì nó không yêu cầu thiết lập kết nối không phải có những cơ chế phức tạp nên tốc độ nhanh...ứng dụng cho các ứng dụng truyền dữ liệu nhanh như chat, game.....



### 3.13.1.3 Port là gì ?

- Port là một tài nguyên logic của máy tính do hệ điều hành quản lý, Port được biểu diễn dưới dạng số và được socket sử dụng. Port và Socket hoạt động trên tầng số 5 (Tầng phiên - Session) để xác định duy nhất một socket của một ứng dụng trên một máy trong mạng. Hay nói cách khác Port là cách mà tầng phiên (Session) phân biệt được giữa các ứng dụng.
- VD: Khi bạn chạy nhiều ứng dụng mạng như Zalo, Skype, Chrome, Mysql, game online... . Ví dụ chương trình Mysql khai báo và sử dụng port 3306. Khi nhận được gói tin trên port 3306 thì hệ điều hành sẽ đưa bản tin này tới socket của Mysql và Mysql sẽ xử lý bản tin này.
- Một TCP/IP Socket gồm một địa chỉ IP kết hợp với một port ? Xác định duy nhất một tiến trình (process ) trên mạng. Hay nói cách khác Đường thông tin trên mạng dựa vào IP là để xác định một máy trên mạng còn port xác định 1 tiến trình trên 1 máy.

### 3.13.2 Tư duy lập trình mạng (Networking) trong Java

- Các ứng dụng phần mềm thường xuyên phải trao đổi dữ liệu với nhau qua môi trường mạng như mạng LAN, WAN hay Internet...
- Với mỗi môi trường khác nhau thì ứng dụng nên có những giao thức mạng phù hợp để quá trình kết nối và trao đổi dữ liệu được thuận tiện và hiệu quả nhất. Một vài ví dụ về giao thức như TCP, UDP, HTTP, Webservice...
- Lập trình mạng là sẽ tạo ra những ứng dụng dạng Client – Server. Tức là sẽ có 2 loại ứng dụng chính đó là Client và Server.
- Quy trình hoạt động của ứng dụng Server – Client như sau:
  - o Server có nhiệm vụ của là lắng nghe, chờ đợi kết nối từ Client trên địa chỉ IP của mình với một PORT xác định nào đó. Khi client gửi dữ liệu tới Server thì nó sẽ nhận được sau đó xử lý và trả lại hồi đáp cho client theo một giao thức nào đó.
  - o Client là ứng dụng được phục vụ, nó chỉ gửi truy vấn và chờ đợi kết quả từ Server

### 3.13.3 Khởi tạo một Socket Server trong java

- Các bước khởi tạo một Socket Server trong java như sau:
  - o Bước 1: Khởi tạo đối tượng ServerSocket với một số hiệu Port
  - o Bước 2: Lắng nghe để chờ đợi kết nối tới
  - o Bước 3: đọc dữ liệu từ InputStream
  - o Bước 4: Đóng kết nối và đóng Server Socket.
- Xét ví dụ sau như bên dưới mô tả một server lắng nghe trên port 10000 và mỗi khi có kết nối tới thì server sẽ đọc dữ liệu String được truyền tới và xuất ra nội dung data String nhận được. Đồng thời sẽ kết thúc server khi nhận được nội dung “bye” từ client hoặc Client đóng kết nối.

```

18  * @author tanth
19  */
20 public class ServerSocketExample {
21     public static void main(String[] args) {
22         try {
23             ServerSocket server = new ServerSocket(10000);
24             System.out.println("Server is running.....");
25             Socket socket = server.accept();
26             DataInputStream input = new DataInputStream(socket.getInputStream());
27             String data = "";
28             while (true) {
29                 data = input.readUTF();
30                 System.out.println("Server Received: " + data);
31                 if(data.equals("bye")){
32                     break;
33                 }
34             }
35             input.close();
36             socket.close();
37             server.close();
38             System.out.println("The server has stopped ....");
39         } catch (EOFException ex) {
40             System.out.println("Client has disconnected");
41         } catch (IOException ex) {
42             System.out.println("Error: "+ex.getMessage());
43         }
44     }
45 }
46

```

- o Tại dòng 23, khai báo biến đối tượng server đồng thời khởi tạo đối tượng kiểu ServerSocket và binding với port 10000.
- o Tại dòng 25, Server bắt đầu lắng nghe các kết nối đến từ client trên port 10000. Chương trình sẽ dừng lại cho tới khi có kết nối từ client tới và nhận về một socket.
- o Tại dòng 26 khai báo dòng nhập xuất high level để nhận dữ liệu đến từ client.
- o Tại dòng 29 đọc từng dòng dữ liệu String dạng Unicode
- o Tại dòng 30 hiển thị dữ liệu nhận được từ client truyền tới
- o Tại dòng 31 kiểm tra nếu nhận được chuỗi ký tự “bye” từ client thì server sẽ chủ động đóng kết nối với client và kết thúc lắng nghe từ server.

- Tại dòng 35,36,37 thực hiện đóng các InputStream, Socket và ServerSocket để giải phóng tài nguyên hệ thống.
- Kết quả khi chạy chương trình như sau:

The screenshot shows the Eclipse IDE's Output window with three tabs: Debugger Console, JavaCoreAdvance (run), and JavaCoreAdvance (run) #2. The JavaCoreAdvance (run) tab is active, displaying the following log output:

```

run:
Server is running.....
Server Received: Hello
Server Received: My name is Alice
Server Received: Nice to meet you?
Server Received: bye
The server has stopped ....

```

### 3.13.4 Khởi tạo một Client Socket trong java

- Các bước khởi tạo một Socket Server trong java như sau:
  - Bước 1: Khởi tạo đối tượng Socket với một số hiệu Port và địa chỉ của Server đang lắng nghe để chờ đợi kết nối tới.
  - Bước 2: Kết nối tới server
  - Bước 3: Gửi dữ liệu tới server.
  - Bước 4: Đóng kết nối khi thực hiện xong nhiệm vụ.
- Sau đây chúng ta xem ví dụ về client gửi các bản tin string cho server như sau:

```

16  * @author tanth
17  */
18  public class ClientSocketExample {
19      public static void main(String[] args) {
20          System.out.println("Client is running....");
21          try {
22              Socket client = new Socket("localhost", 10000);
23              DataOutputStream Output =
24                  new DataOutputStream(client.getOutputStream());
25              Output.writeUTF("Hello");
26              Output.writeUTF("My name is Alice");
27              Output.writeUTF("Nice to meet you?");
28              Output.writeUTF("bye");
29              Output.flush();
30              Output.close();
31              client.close();
32              System.out.println("Client has said bye....");
33          } catch (IOException ex) {
34              System.out.println("Error: " + ex.getMessage());
35          }
36      }
37  }

```

- Tại dòng 22 chúng ta khai báo một biến đối tượng client có kiểu dữ liệu là Socket và khởi tạo đối tượng Socket để kết nối tới server có địa chỉ “localhost” ở port 10000. Tức là kết nối tới chính server trên cùng máy.
- Tại Dòng 23 khai báo và khởi tạo một DataOutputStream để xuất dữ liệu và truyền sang cho server.
- Từ dòng 25 đến dòng 28 thực hiện ghi các dữ liệu String lên dòng xuất dữ liệu. các dữ liệu này sẽ được truyền sang cho server ngay khi gọi lệnh flush tại dòng 29.
- Tại dòng 26,27 chúng ta thực hiện đóng Stream và kết nối ngay sau khi truyền xong dữ liệu tới server.
- Dưới đây là kết quả sau khi chạy chương trình:

```
JavaCoreAdvance (run) X JavaCoreAdvance (run) #2 X
run:
Client is running....
Client has said bye....
```

- Nếu không kết nối được tới server vì một lý do nào đó như server chưa bật, hoặc network bị lỗi thì client sẽ nhận được ConnectException sau một khoảng thời gian timeout như kết quả dưới đây:

```
JavaCoreAdvance (debug) X Debugger Console X
debug:
Client is running....
Error: Connection refused: connect
```

### 3.13.5 Cơ chế 1 Server phục vụ nhiều Client

- Để đảm bảo ứng dụng server có thể cùng lúc phục vụ được nhiều client cùng lúc thì ứng dụng server sẽ phải thực hiện được các việc sau:
  - Có thể Accept nhiều connection đồng thời.
  - Với mỗi kết nối từ client sẽ được ứng dụng server xử lý ở một thread độc lập.
- Đoạn mã dưới đây đã sử dụng Thread pool để giúp tách bạch việc Server Accept các kết nối và mỗi khi có kết nối tới thì thread pool sẽ tiếp tục đảm nhiệm xử lý từng kết nối đảm bảo xử lý đồng thời các nhiệm vụ truyền nhận dữ liệu từ client truyền tới.

```

20  * @author tanth
21  */
22  public class ServerSocketMultiClientExample {
23      public static void main(String[] args) {
24          try {
25              ServerSocket server = new ServerSocket(10000);
26              ExecutorService executorService = Executors.newFixedThreadPool(20);
27              System.out.println("Server is running.....");
28              Socket socket = null;
29              while ((socket = server.accept()) != null) {
30                  String clientAddress = socket.getRemoteSocketAddress().toString();
31                  System.out.println("New Connection from " + clientAddress);
32                  executorService.submit(new ServerThread(socket));
33              }
34              executorService.shutdown();
35          } catch (EOFException ex) {
36              System.out.println("Client has disconnected");
37          } catch (IOException ex) {
38              System.out.println("Error: " + ex.getMessage());
39          }
40      }
41  }

16  public class ServerThread implements Callable<Socket> {
17      private Socket socket;
18      ServerThread(Socket socket) {
19          this.socket = socket;
20      }
21      @Override
22      public Socket call() throws Exception {
23          DataInputStream input = new DataInputStream(socket.getInputStream());
24          String data = "";
25          while (true) {
26              data = input.readUTF();
27              System.out.println("Server Received: " + data);
28              if (data.equals("bye")) {
29                  break;
30              }
31          }
32          input.close();
33          socket.close();
34          return socket;
35      }
36  }

```

### 3.13.6 Truyền nhận dữ liệu Client-server socket

### 3.13.7 Truyền nhận một Object giữa client và server

### 3.13.8 URL class trong java

- URL là từ viết tắt của "Uniform Resource Locator". URL được sử dụng để tham chiếu tới tài nguyên trên mạng Internet. URL tạo nên khả năng siêu liên kết cho các website. Mỗi tài nguyên khác nhau lưu trữ trên Internet được gán bằng một địa chỉ chính xác, địa chỉ đó chính là URL. Ví dụ sau là một URL:  
<https://www.google.com:443/maps>
- Trong một URL gồm có các thành phần sau:
  - o Giao thức kết nối (Protocol) : http, https, ftp...
  - o Host hoặc domain Name, ví dụ: google.com
  - o Port: cổng kết nối ví dụ: 443
  - o File: đường dẫn file: ví dụ: maps
  - o Query: các truy vấn đặt trên URL
- URL class là một class trong bộ java.net, nó cung cấp các phương thức cho phép lấy ra các thông tin trong một URL.

```

16  * @author tanth
17  */
18 public class URLExample {
19
20     public static void main(String[] args) throws IOException {
21         try {
22             URL url = new URL("https://www.google.com:443/maps/?user=Alice");
23             System.out.println("Protocol: " + url.getProtocol());
24             System.out.println("Host Name: " + url.getHost());
25             System.out.println("Port Number: " + url.getPort());
26             System.out.println("Default Port: " + url.getDefaultPort());
27             System.out.println("File Name: " + url.getFile());
28             System.out.println("path Name: " + url.getPath());
29             System.out.println("Query: " + url.getQuery());
30         } catch (MalformedURLException ex) {
31             System.out.println("Error: " + ex.getMessage());
32         }
33     }
34 }
35

```

- Kết quả chạy đoạn mã trên như sau:

The screenshot shows the Java IDE's Output window with the title bar 'Output' and tabs for 'Debugger Console' and 'JavaCoreAdvance (run)'. The output content is as follows:

```

run:
Protocol: https
Host Name: www.google.com
Port Number: 443
Default Port: 443
File Name: /maps/?user=Alice
path Name: /maps/
Query: user=Alice

```

### 3.13.9 URLConnection class trong java

- Là một class trong package java.net, có nhiệm vụ tạo kết nối tới server URL, và có thể dùng để đọc, ghi dữ liệu từ server URL.
- Ví dụ bên dưới là một ứng dụng để đọc dữ liệu từ một trang web trả về và hiển thị trên màn hình console.

```

19 |     * @author tanth
20 |     */
21 | public class URLConnectionExample {
22 |
23 |     public static void main(String[] args) {
24 |         try {
25 |             URL url = new URL("http://activestudy.online");
26 |             URLConnection urlConnection = url.openConnection();
27 |             DataInputStream input =
28 |                 new DataInputStream(urlConnection.getInputStream());
29 |             String data = "";
30 |             while (input.available() > 0) {
31 |                 data = input.readUTF();
32 |                 System.out.println("Data: " + data);
33 |             }
34 |         } catch (IOException ex) {
35 |             System.out.println("Error: " + ex.getMessage());
36 |         }
37 |     }
38 |
39 }

```

### 3.13.10 HttpURLConnection class trong java

- Là một class trong package java.net, HttpURLConnection class kế thừa từ URLConnection và có nhiệm vụ tạo kết nối tới http server URL và chỉ dành cho giao thức http mà thôi.
- Ví dụ dưới đây là cách dùng HttpURLConnection để truy cập các thông tin header, request, response và data trả về từ kết nối http tới URL server.

```

20  * @author tanth
21  */
22  public class HttpURLConnectionExample1 {
23      public static void main(String[] args) {
24          try {
25              URL url = new URL("http://activestudy.online");
26              HttpURLConnection httpURLConnection =
27                  (HttpURLConnection) url.openConnection();
28              for (int i = 1; i <= 8; i++) {
29                  System.out.println(httpURLConnection.getHeaderFieldKey(i) +
30                      " = " + httpURLConnection.getHeaderField(i));
31              }
32              String logExport =
33                  "Method: " + httpURLConnection.getRequestMethod() +
34                  "Content Type" + httpURLConnection.getContentType() +
35                  "Response code: " + httpURLConnection.getResponseCode() +
36                  "Response message:" + httpURLConnection.getResponseMessage();
37              System.out.println(logExport);
38
39              DataInputStream input =
40                  new DataInputStream(httpURLConnection.getInputStream());
41              String data = "";
42              while (input.available() > 0) {
43                  data = input.readUTF();
44                  System.out.println("Data: " + data);
45              }
46              httpURLConnection.disconnect();
47          } catch (IOException ex) {
48              System.out.println("Error: " + ex.getMessage());
49          }
50      }
51  }

```

Kết quả chạy chương trình như sau:

Output

Debugger Console X JavaCoreAdvance (run) X

```

run:
Server = nginx/1.10.2
Date = Wed, 21 Nov 2018 05:26:41 GMT
Content-Type = text/html; charset=utf-8
Content-Length = 676745
Connection = keep-alive
Vary = Accept-Encoding
X-Powered-By = Express
x-cache = HIT
Response message:OK
Data: DOCTYPE html><html><head><meta charset="utf-8" class="next-head
*{border-width:0;box-sizing:border-box;margin:0;padding:0;-webkit-text
.elementHidden{visibility:hidden;}</style><style id="__jsx-1208982534"
div.widget-timer.jsx-2226191955 .timer.jsx-2226191955{font-size:54px;}
div.widget-timer.jsx-2226191955 .timer-labels.jsx-2226191955{position:
div.widget-timer.jsx-2226191955 .timer-labels.jsx-2226191955 .timer-la
div.widget-timer.jsx-2226191955 .timer-labels.jsx-2226191955 .timer-la
div.widget-timer.jsx-2226191955 .timer-labels.jsx-2226191955 .timer-la

```

### 3.13.11 InetAddress class trong java

- Class InetAddress nằm trong package java.net, được sử dụng để lấy địa chỉ IP của một host.
- Đối tượng InetAddress không thể được khai báo trực tiếp mà có thể đạt được đối tượng này thông qua một số các static method sau:

● <b>getAllByName</b> (String host)	InetAddress[]
● <b>getByAddress</b> (byte[] addr)	InetAddress
● <b>getByAddress</b> (String host, byte[] addr)	InetAddress
● <b>getByName</b> (String host)	InetAddress
● <b>getLocalHost()</b>	InetAddress
● <b>getLoopbackAddress()</b>	InetAddress

- Sau đây là ví dụ về khai báo và sử dụng InetAddress để đạt được địa chỉ IP của một host.

```

15  * @author tanth
16  */
17  public class InetAddressExample {
18      public static void main(String[] args) {
19          try {
20              InetAddress ipAddress = InetAddress.getByName("activestudy.online");
21              System.out.println("IP: " + ipAddress.getHostAddress());
22              System.out.println("Domain: " + ipAddress.getHostName());
23          } catch (UnknownHostException ex) {
24              System.out.println("Error: " + ex.getMessage());
25          }
26      }
27  }
28

```

### 3.13.12 DatagramSocket and DatagramPacket trong Java

- DatagramSocket và DatagramPacket là 2 class trong bộ package java.net. Chúng giúp truyền nhận dữ liệu theo giao thức UDP bằng một connection-less. Hay nói cách khác là truyền nhận dữ liệu điểm điểm mà không cần duy trì kết nối trước.
- Ví dụ dưới đây là cách tạo ra một điểm truyền và một điểm nhận dữ liệu thông qua UDP protocol.

```

19  * @author tanth
20  */
21  public class DatagramSocketExample {
22      public static void main(String[] args) {
23          InetAddress remoteIP;
24          try (DatagramSocket socket = new DatagramSocket()) {
25              remoteIP = InetAddress.getLocalHost();
26              int remotePort = 10001;
27              String data = "Hello, My name Tanth";
28              DatagramPacket dataPackage =
29                  new DatagramPacket(data.getBytes(), data.length(), remoteIP, remotePort);
30              socket.send(dataPackage);
31              System.out.println("The data was successfully sent");
32          } catch (IOException ex) {
33              System.out.println("Error: " + ex.getMessage());
34          }
35      }
36  }
37
18  * @author tanth
19  */
20  public class DatagramSocketReceiveExample {
21      public static void main(String[] args) {
22          System.out.println("UDP server is starting...");
23          int localPort = 10001;
24          try(DatagramSocket socket = new DatagramSocket(localPort)) {
25              final int bufferSize = 1024;
26              byte[] buffer = new byte[bufferSize];
27              DatagramPacket dataPackage = new DatagramPacket(buffer, bufferSize);
28              System.out.println("UDP server is listening on port: " + localPort );
29              socket.receive(dataPackage);
30
31              String data = new String(dataPackage.getData(),
32                                      0, dataPackage.getLength());
33
34              System.out.println("Received Data: " + data);
35          } catch (IOException ex) {
36              System.out.println("Error: " + ex.getMessage());
37          }
38      }
39  }

```

Dưới đây là kết quả chạy chương trình

```
Output
Debugger Console X JavaCoreAdvance (run) X JavaCoreAdvance (run) #2 X
run:
UDP server is starting...
UDP server is listening on port: 10001
Received Data: Hello, My name Tanh
```

### 3.14 Java và Database:

#### 3.14.1 Tư duy lập trình giao tiếp Database

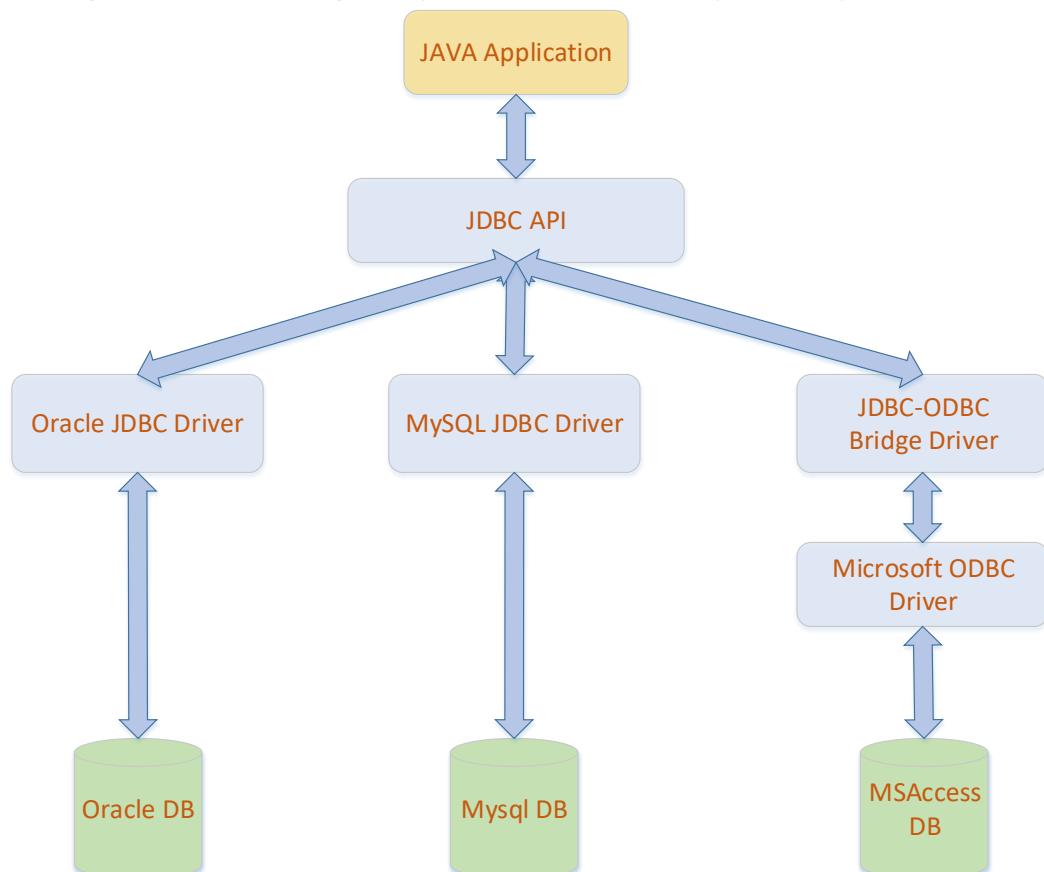
#### 3.14.2 Cở sở dữ liệu quan hệ (RDB)

#### 3.14.3 Cở sở dữ liệu Không hệ (NoRDB)

#### 3.14.4 Truy vấn cơ sở dữ liệu quan hệ (SQL)

#### 3.14.5 JDBC

- JDBC là Java API để truy cập tới cơ sở dữ liệu quan hệ (RDB)
- JDBC cung cấp giao diện lập trình đồng bộ cho chương trình Java có thể truy cập tới nhiều loại cơ sở dữ liệu quan hệ(Oracle, MSQL, MySQL...), trong một hệ thống tập trung hoặc phân tán(Client server), thậm chí là môi trường không đồng nhất(Window, linux, Readhat, Centos...)
- JDBC API, JDBC Driver và Relational Database khác nhau. Khi lập trình chúng ta sử dụng JDBC API, JDBC API sử dụng các JDBC Driver để giao tiếp với các Database. Mỗi quan hệ này được mô tả như ở hình bên dưới.



- Sau đây chúng ta sẽ xét một ví dụ về việc truy cập MySql Database sử dụng JDBC như sau:

```

19  * @author tanth
20  */
21  public class JDBCExample {
22      public static void main(String[] args) {
23          try {
24              Class.forName("com.mysql.jdbc.Driver");
25
26              System.out.println("Driver loaded");
27              String url = "jdbc:mysql://localhost/activestudy";
28              String userName = "ActiveStudyUser";
29              String passWord = "qwertyuiop";
30
31              Connection connection = DriverManager.getConnection
32                  (url, userName , passWord );
33              Statement statement = connection.createStatement();
34              String strQuery = "SELECT StudentID, Name, BirthDay, StudentCode "
35                  + "FROM student";
36              ResultSet resultSet = statement.executeQuery(strQuery);
37              ArrayList<Student> students = new ArrayList<>();
38
39              while (resultSet.next()){
40                  Student student = new Student();
41                  student.setStudentID(resultSet.getString(1));
42                  student.setName(resultSet.getString(2));
43                  student.setBirthday(resultSet.getString(3));
44                  student.setStudentCode(resultSet.getString(4));
45                  System.out.println("Name: "+ student.getName());
46                  System.out.println("Code: "+ student.getStudentCode());
47                  System.out.println("ID : "+ student.getStudentID());
48                  System.out.println("BirthDay: "+ student.getBirthday());
49                  System.out.println("-----");
50              }
51              connection.close();
52          } catch (ClassNotFoundException|SQLException ex) {
53              /*Log here */
54          }
55      }
56  }

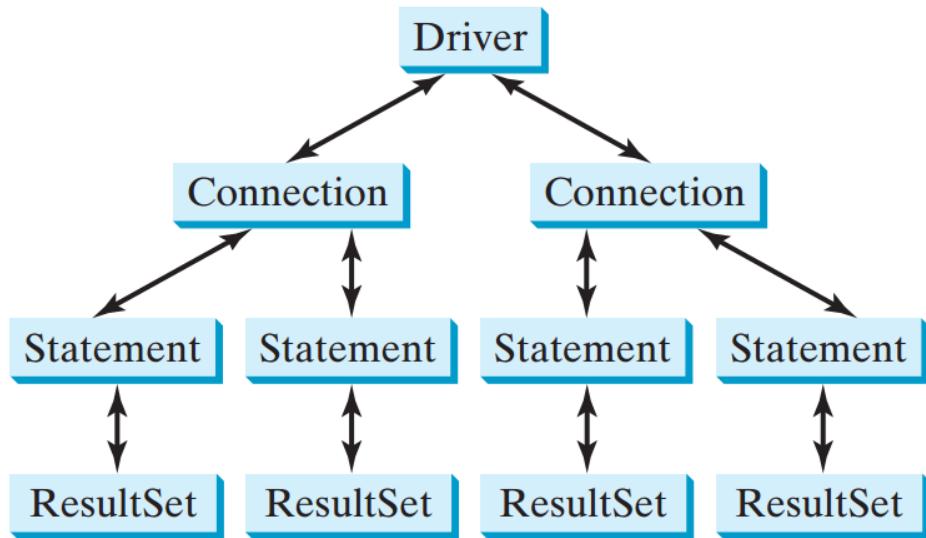
```

Bên dưới đây là kết quả sau khi thực hiện chương trình.

```
Output
Debugger Console X JavaCoreAdvance (run) X

run:
Driver loaded
Name: Hoàng Nam
Code: SV10001
ID : 1
BirthDay: 1997-07-01
-----
Name: Nguyễn Hoàng Mạnh Dũng
Code: SV10002
ID : 2
BirthDay: 1997-12-05
```

Dưới đây là các bước để thực hiện truy cập tới Database:



### 3.14.5.1 Loading drivers.

- Driver Database tương ứng cần phải được load lên trước khi tạo ra kết nối tới Database.

24

```
Class.forName("com.mysql.jdbc.Driver");
```

Database	Driver Class	Source	Link download
MySQL	com.mysql.jdbc.Driver	mysql-connector-java-8.0.13.jar	<a href="https://dev.mysql.com/downloads/file/?id=480292">https://dev.mysql.com/downloads/file/?id=480292</a>
MariaDB	jdbc:mariadb	mariadb-java-client-2.3.0.jar	<a href="https://downloads.mariadb.com/Connectors/java/connector-java-2.3.0/mariadb-java-client-2.3.0.jar">https://downloads.mariadb.com/Connectors/java/connector-java-2.3.0/mariadb-java-client-2.3.0.jar</a>

Oracle	oracle.jdbc.driver.OracleDriver	ojdbc7.jar	<a href="https://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html">https://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html</a>
Mssql	jdbc:sqlserver	Microsoft JDBC Driver 7.0 for SQL Server	<a href="https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-2017">https://docs.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-2017</a>
Access	sun.jdbc.odbc.JdbcOdbcDriver	Tích hợp sẵn trong bộ JDK	

- Sau khi download về chúng ta có thể phải cài đặt hoặc đổi với Oracle, Mysql, Mariadb chúng ta sẽ thêm đường dẫn thư viện vào classpath

```
set classpath=%classpath%;d:\lib\mysql-connector-java-8.0.13.jar;
d:\lib\ojdbc7.jar
```

- Hoặc có thể add trực tiếp thư viện vào IDE để phát triển ứng dụng Java.

### 3.14.5.2 Establishing connections

- Trước khi truy vấn cơ sở dữ liệu thì chúng ta cần phải tạo kết nối từ ứng dụng của chúng ta tới Database. và giữ kết nối cho tới khi không cần truy vấn dữ liệu nữa hoặc ngắt chương trình.

```
27 |           String url = "jdbc:mysql://localhost/activestudy";
28 |           String userName = "ActiveStudyUser";
29 |           String passWord = "qwertyuiop";
30 |           Connection connection = DriverManager.getConnection
31 |                               (url, userName , passWord );
```

- Bằng cách gọi tới method getConnection() để tạo kết nối tới database. Trước đó Database cần phải được start service lên trên máy có địa chỉ IP được chỉ ra trong url. Trong ví dụ trên địa chỉ Database là máy localhost. Nếu chạy trên máy khác thì cần phải được thay thế địa chỉ này trong url.
- url String là chuỗi có thể chứa những thành phần: driver name, IP database server, Database name, Username, Password và một số tham số khác. Trong ví dụ trên url không chứa Username và password. lúc này Username và password được truyền độc lập vào method getConnection().
- Đối tượng Connection tạo ra để duy trì kết nối ảo tới database server. Mọi Truy vấn sẽ thông qua kết nối này.
- Khi tạo kết nối nếu Database server chưa được start service hoặc thời gian quá lâu thì chương trình java sẽ nhận được một SQLException. Lúc này chúng ta cần kiểm tra lại url hoặc service của Database server đã bật hay chưa.

### 3.14.5.3 Creating statements

- Nếu một Connection có thể được hình dung như một sợi dây cáp để liên kết chương trình của chúng ta với cơ sở dữ liệu, thì một đối tượng Statement có thể được xem như một giỏ hàng cung cấp các câu lệnh SQL để thực thi bởi cơ sở dữ liệu và đưa kết quả trả lại chương trình. Khi một đối tượng kết nối được tạo, chúng ta có thể tạo các câu lệnh để thực thi các câu lệnh SQL như sau:

32

```
Statement statement = connection.createStatement();
```

### 3.14.5.4 Executing statements.

- Sau khi có Statement chúng ta đi thực hiện truy vấn dữ liệu tới Database thông qua SQL data definition language (DDL) và một trong hai method sau đây của đối tượng statement: executeQuery() hoặc executeUpdate().

```
33
34     String strQuery = "SELECT StudentID, Name, BirthDay, StudentCode "
35             + "FROM student";
36     ResultSet resultSet = statement.executeQuery(strQuery);
37
38     String strQueryUpdate = "Update Name = 'Đúc Thái' from student "
39             + "where StudentID = 1";
40     statement.executeUpdate(strQueryUpdate);
```

Sự khác biệt giữa hai method này ở chỗ mục đích và kết quả trả về.

### 3.14.5.5 Processing ResultSet

- ResultSet được hình dung như một danh sách các bản ghi được trả về từ câu lệnh truy vấn Database và bản ghi đầu tiên là một bản ghi null.
- ResultSet cung cấp các phương thức để truy cập lần lượt tới các bản ghi thông qua method next().
- Khi gọi tới phương thức next() nghĩa là chúng ta đang muốn di chuyển tới bản ghi tiếp theo để truy cập tới bản ghi đó
- Chúng ta gọi method next() liên tục để di chuyển tới các bản ghi kế tiếp, nếu vẫn còn bản ghi kế tiếp thì phương thức next() trả về giá trị true. Nếu tới bản ghi cuối cùng thì method next() này sẽ trả về một giá trị false, và lúc này là lúc chúng ta đã truy cập lần lượt tới toàn bộ các bản ghi được trả về từ câu lệnh truy vấn SQL DDL

```
41
42     ArrayList<Student> students = new ArrayList<>();
43     while (resultSet.next()) {
44         Student student = new Student();
45         student.setStudentID(resultSet.getString(1));
46         student.setName(resultSet.getString(2));
47         student.setBirthDay(resultSet.getString(3));
48         student.setStudentCode(resultSet.getString(4));
49         System.out.println("Name: " + student.getName());
50         System.out.println("Code: " + student.getStudentCode());
51         System.out.println("ID : " + student.getStudentID());
52         System.out.println("BirthDay: " + student.getBirthDay());
53         System.out.println("-----");
54     }
55     connection.close();
56 } catch (ClassNotFoundException|SQLException ex) {
57     /*Log here */
58 }
```

- ResultSet còn cung cấp các method để lấy ra dữ liệu ở bản ghi hiện tại như ví dụ tại dòng 45 tới 48.

```
resultSet.getString(2));
```

getString là method để lấy ra một giá trị kiểu String tại vị trí (cột) thứ hai của bản ghi hiện tại(bản ghi mà resultSet đang focus tới). Cột đầu tiên có số thứ tự là 1.

- Ngoài ra ResultSet còn cung cấp nhiều method khác để lấy ra các loại dữ liệu khác nhau tùy theo thứ tự các field sắp xếp theo câu truy vấn.

● <b>getCursorName()</b>	String ^
● <b>getDate(String columnLabel)</b>	Date
● <b>getDate(int columnIndex)</b>	Date
● <b>getDate(String columnLabel, Calendar cal)</b>	Date
● <b>getDate(int columnIndex, Calendar cal)</b>	Date
● <b>getDouble(String columnLabel)</b>	double
● <b>getDouble(int columnIndex)</b>	double
● <b>getFetchDirection()</b>	int
● <b>getFetchSize()</b>	int
● <b>getFloat(String columnLabel)</b>	float
● <b>getFloat(int columnIndex)</b>	float
● <b>getHoldability()</b>	int
● <b>getInt(String columnLabel)</b>	int
● <b>getInt(int columnIndex)</b>	int
● <b>getLong(String columnLabel)</b>	long
● <b>getLong(int columnIndex)</b>	long
● <b>getMetaData()</b>	ResultSetMetaData v

Instance Members; Press 'Ctrl+SPACE' Again for All Items

### 3.14.6 PreparedStatement

- Statement là một đối tượng để thực hiện các câu truy vấn tĩnh.
- PreparedStatement là đối tượng được extend từ statement, nó có khả năng cho phép truyền vào các parameter để xây dựng các câu truy vấn động, tạo khả năng mềm dẻo khi truy vấn cơ sở dữ liệu.

```

20  * @author tanth
21  */
22  public class PreparedStatementExample {
23
24      public static void main(String[] args) {
25          try {
26              Class.forName("com.mysql.jdbc.Driver");
27
28              System.out.println("Driver loaded");
29              String url = "jdbc:mysql://localhost/activestudy";
30              String userName = "ActiveStudyUser";
31              String passWord = "qwertyuiop";
32              Connection connection = DriverManager.getConnection(url, userName, passWord);
33              PreparedStatement preparedStatement = connection.prepareStatement(
34                  "insert into Student (StudentID, Name, BirthDay, StudentCode) "
35                  + "values (?, ?, ?, ?)");
36              preparedStatement.setString(1, "3");
37              preparedStatement.setString(2, "Thắng");
38              preparedStatement.setString(3, "1997-09-05");
39              preparedStatement.setString(4, "SV003");
40
41              ResultSet resultSet = preparedStatement.executeQuery();
42
43              connection.close();
44          } catch (ClassNotFoundException | SQLException ex) {
45              /*Log here */
46          }
47      }
48  }

```

### 3.14.7 CallableStatement

- CallableStatement là một interface cung cấp khả năng cho phép statement thực thi một SQL-stored procedures đã được tạo ra ở database.
- Các stored procedures cần phải truyền vào các tham số IN, OUT, INOUT và có thể return ra các giá trị

## 3.15 API và Webservice trong Java

### 3.15.1 Hiểu biết về API

- API là thuật ngữ viết tắt của Application Programming Interface (Giao diện lập trình ứng dụng).
- Mục đích chính của một API là cung cấp khả năng truy xuất đến một tập các hàm dành cho các module hoặc chương trình khác truy xuất tới
- API được các nhà phát triển chương trình phần mềm định nghĩa và công bố, vì vậy nó thường được công bố dưới hình thức là các Interface. Chương trình muốn mở API thì thực thi (Implement) interface đó.
- API thường được định nghĩa là các Interface vì vậy mỗi bộ API thường là tập hợp các method có tên, tham số truyền vào(Parameter) và kiểu giá trị trả ra (Return value).
- API được xây dựng dựa trên các giao thức truy cập, một trong số đó là các giao thức và kiến trúc như: Restful, SOAP, RMI, TCP/IP, HTML, XML, XML RPC, DLL,...

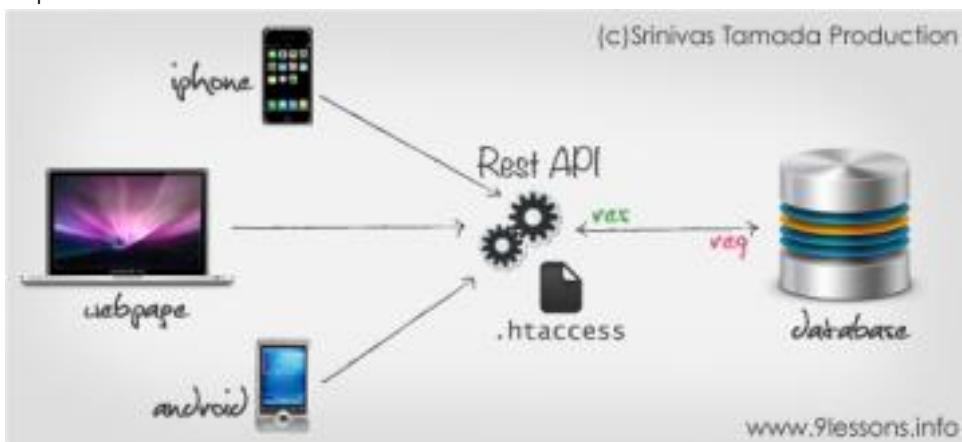
### 3.15.2 Hiểu biết về webservice

- Webservice là một hình thức mở kết nối API của ứng dụng.
- Webservice sử dụng các giao thức dựa trên giao thức cơ sở là HTTP.
- Các giao thức webservice phổ biến nhất hiện nay đó là Restful và SOAP, ngoài ra còn có XML RPC

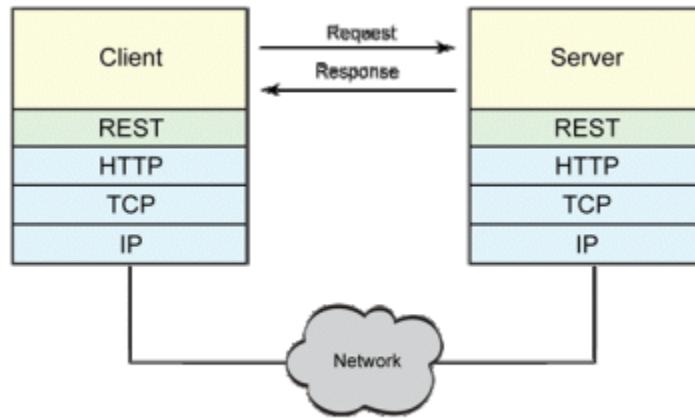
- Webservice là một dịch vụ web, nó là một khái niệm rộng hơn so với khái niệm web thông thường, nó được sử dụng bởi các ứng dụng, nó cung cấp các thông tin, dữ liệu thô, và khó hiểu với đa số người dùng. Các ứng dụng này sẽ chế biến các dữ liệu thô trước khi trả về cho người dùng cuối cùng thông qua các ứng dụng Web, app khác thân thiện hơn với người dùng.
- Dịch vụ Web (Web Service) là một ứng dụng tuyệt vời của kỹ thuật lập trình phân tán và được coi là một công nghệ mang đến cuộc cách mạng trong khả năng cung cấp các dịch vụ của các hệ thống ứng dụng phần mềm.
- Các phương thức mở API truyền thống từ trước hoạt động tốt trong môi trường truy cập trong phạm vi cùng một máy tính, hoặc trong một mạng LAN. Tuy nhiên ngày nay đòi hỏi các ứng dụng chạy trên cloud, và giao tiếp qua môi trường internet, chính vì vậy các giao thức API cũ đã không thể đáp ứng cho nhu cầu này.
- HTTP là một giao thức hoạt động tốt trong môi trường Internet, nó có khả năng đặc quyền được vượt qua hầu hết các firewall mà không cần thiết phải mở port tại các node mạng. Chính điều này HTTP đã được chọn làm giao thức cơ sở cho các giao thức Webservice ra đời và phát triển.

### 3.15.3 RESTful webservice

- RESTful Web Service là các Web Service được viết dựa trên kiến trúc REST . REST là từ viết tắt của Representational State Transfer.



- Nói đến RESTful Webservice là người ta nói đến chuẩn công nghệ để tạo ra một Webservice. Khác với SOAP thì RESTful xây dựng một webservice có kiến trúc truy nhập theo hướng tài nguyên (Resource) và dễ dàng sử dụng hơn nhiều. RESTful giờ đây dần thay thế hoàn toàn SOAP để hướng đến trở thành một chuẩn công nghệ webservice phổ biến nhất hiện nay, được các ông lớn về công nghệ ủng hộ và sử dụng như Facebook, google..
- RESTful hoạt động ở phía trên của chั้ng giao thức HTTP. Giao tiếp giữa Client và webservice Server dựa trên Request và Response HTTP.



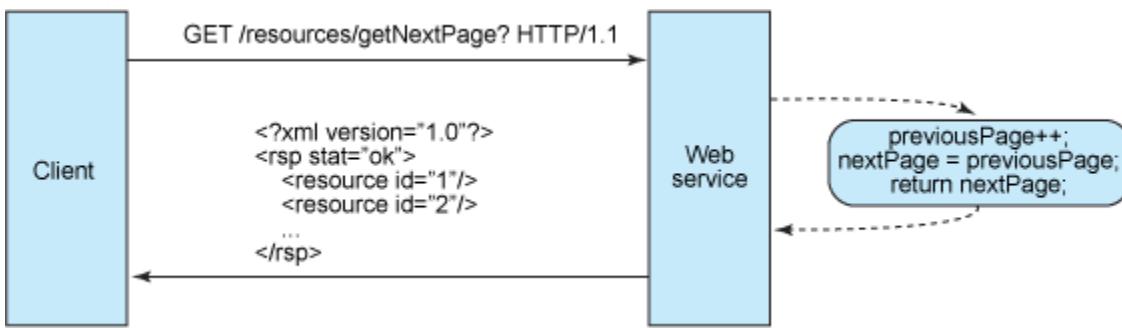
- RESTful được thiết kế dựa trên 4 nguyên tắc như sau:
  - o Sử dụng các phương thức của HTTP một cách rõ ràng (GET, PUT, POST, DELETE)
  - o Phi trạng thái
  - o Hiển thị tài nguyên như dưới dạng cấu trúc thư mục như URLs
  - o Hỗ trợ truyền tải dữ liệu với JSON (JavaScript Object Notation) và XML hoặc cả hai.
- Sau đây chúng ta đi tìm hiểu về các nguyên tắc của RESTful để hiểu rõ hơn

#### 3.15.3.1 RESTful Sử dụng các phương thức của HTTP một cách rõ ràng (GET, PUT, POST, DELETE)

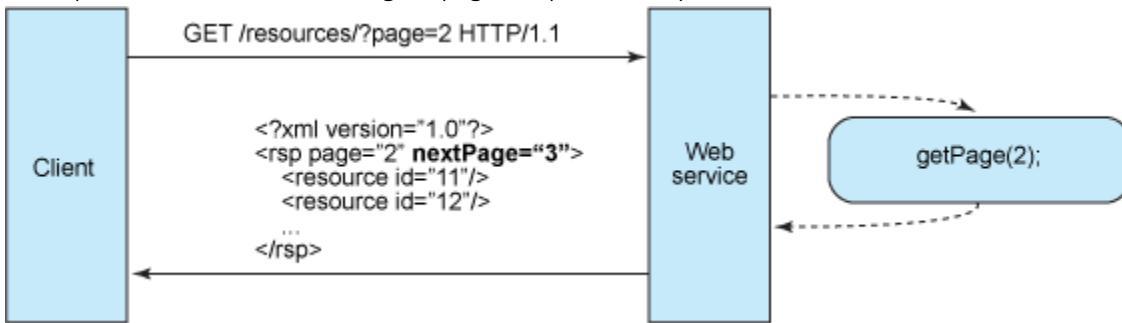
- Với chuẩn RESTful cho phép và khuyến cáo chúng ta sử dụng các phương thức (HTTP method) một cách rõ ràng đúng chức năng về mặt ý nghĩa của nó và cụ thể là như sau:
  - o GET: Được dùng để truy vấn một tài nguyên, thông tin (tương đương với lệnh SELECT).
  - o POST : Được dùng để tạo ra một dữ liệu, tài nguyên mới trên máy chủ (Tương đương với lệnh INSERT).
  - o PUT : Được dùng để thay đổi trạng thái một tài nguyên hoặc để cập nhật (Tương đương UPDATE).
  - o DELETE: Được dùng để huỷ bỏ hoặc xoá một dữ liệu, tài nguyên ,(Tương đương DELETE).

#### 3.15.3.2 Phi trạng thái

- Không giống như các dịch vụ web thông thường các máy chủ web lưu giữ lại trạng thái của phiên làm việc (session) để nâng cao hiệu quả truy xuất trong quá trình request nội dung giữa các page. Điều này được yêu cầu chặt chẽ trong J2EE và rất quen thuộc với những ai phát triển website. Với một Restful webservice thì việc lưu trữ trạng thái là việc của ứng dụng phía Client. Cũng chính vì vậy mà việc phân bổ request tới các server để cân bằng tải (Load balance) được thực hiện với webservice một cách thuận tiện và không vấp phải vướng mắc gì.
- VD Thiết kế theo trạng thái. Trạng thái của Page được phía server lưu trữ lại để phục vụ cho lần request tiếp theo



- VD Thiết kế phi trạng thái. Trạng thái của Page được phía client chủ động lưu trữ lại cho lần request tiếp theo. phía server chỉ trả về thông tin page mà phía client yêu cầu.



### 3.15.3.3 Hiển thị tài nguyên như dưới dạng cấu trúc thư mục như URLs

- Như đặc tính của Restful nhằm xây dựng một dịch vụ hướng tới tài nguyên của hệ thống. Tài nguyên của hệ thống được tổ chức thành cây phân nhánh, phân lớp, có nhánh cha nhánh con tựa như một cây thư mục, và để truy cập đến các tài nguyên này thì URI cũng được tổ chức như vậy để trực quan và dễ hiểu khi truy cập tới các tài nguyên này.

ví dụ về URI tài nguyên:

<http://activestudy.online/khoa hoc/lop hoc/hoc vien/{mahocvien}>

Vậy là nhìn vào URI mà một client có thể sử dụng để truy cập tới một tài nguyên Học viên “hocvien” như thế nào. Đầu tiên là URL truy cập tới dịch vụ webservice <http://activestudy.online/>. Tiếp theo ta có các tài nguyên “khoa hoc” trong tài nguyên khóa học có các tài nguyên “lop hoc”. Trong tài nguyên lớp học có tài nguyên “hocvien” và khi truy cập tới tài nguyên học viên thì ta có thể chỉ ra chính xác học viên nào sẽ được truy cập dựa vào mã học viên “{mahocvien}”.

Vậy các tài nguyên này thực sự là gì, trong Java chúng ta sẽ tổ chức các tài nguyên này dưới dạng một “class” để đáp ứng cho tài nguyên. Ví dụ để mô phỏng cho tài nguyên này thì ta có các class :KhoaHoc, lopHoc, hocVien. Ta sẽ đi sâu vào vấn đề này trong phần tiếp theo đó là xây dựng một webservice theo chuẩn restful.

### 3.15.3.4 Hỗ trợ truyền tải dữ liệu với JSON (JavaScript Object Notation) và XML hoặc XHTML.

- Khi dữ liệu được truyền tải giữa rest client và rest server thì cần phải chỉ rõ loại hình tổ chức dữ liệu được sử dụng, kiểu này được chỉ rõ ở thuộc tính “Content-Type” của HTTP header. có các kiểu tổ chức dữ liệu như sau: **XML, JSON, XHTML**.

### 3.15.3.5 Ví dụ về một chương trình RESTful API phục vụ login, Logout và truy vấn thông tin user.

- Định nghĩa một Interface IAuthentication, Bao gồm 3 method phục vụ login, logout và lấy thông tin user.

```

8  import javax.ws.rs.core.Response;
9
10 /**
11 *
12 * @author tanth
13 */
14 public interface IAuthentication {
15     public Response login(String content);
16     public Response logout(String content);
17     public Response getUser(String user);
18 }
```

- Implement interface IAuthentication và thêm vào các Anotation theo chuẩn của REST để ứng dụng nhận ra các thành phần Path của các resource.

```

8  import java.util.logging.Level;
9  import java.util.logging.Logger;
10 import javax.ws.rs.GET;
11 import javax.ws.rs.POST;
12 import javax.ws.rs.Path;
13 import javax.ws.rs.Produces;
14 import javax.ws.rs.QueryParam;
15 import javax.ws.rs.core.MediaType;
16 import javax.ws.rs.core.Response;
17 import org.codehaus.jettison.json.JSONException;
18 import org.codehaus.jettison.json.JSONObject;
19
20 /**
21 *
22 * @author tanth
23 */
```

```
24     @Path("/rest")
25     public class MyAPIService implements IAuthentication {
26         @POST
27         @Produces(MediaType.APPLICATION_JSON)
28         @Path("/login")
29         @Override
30         public Response login(String content) {
31             JSONObject result = new JSONObject();
32             try {
33                 JSONObject loginObj = new JSONObject(content);
34                 String user = loginObj.getString("user");
35                 String password = loginObj.getString("password");
36                 if (user.equals("activeStudy") && password.equals("123456")) {
37                     result.put("code", "0");
38                     result.put("explain", "Success");
39                 } else {
40                     result.put("code", "1");
41                     result.put("explain", "UnSuccess");
42                 }
43             } catch (JSONException ex) {
44                 return Response.status(500).build();
45             }
46             return Response.status(200).entity(result.toString()).build();
47         }
}
```

```

49      @POST
50      @Produces(MediaType.APPLICATION_JSON)
51      @Path("/logout")
52      @Override
53      public Response logout(String content) {
54          return Response.status(200).build();
55      }
56
57      //-----
58      @GET
59      @Produces(MediaType.TEXT_PLAIN)
60      @Path("/getUser")
61      @Override
62      public Response getUser(@QueryParam("user") String user) {
63          JSONObject result = new JSONObject();
64          try {
65              result.put("user", user);
66              result.put("Name", "Nguyễn Hoàng");
67              result.put("register", "21/12/2018");
68          } catch (JSONException ex) {
69              Logger.getLogger(MyAPIService.class.getName()).log(Level.SEVERE, null,
70              return Response.status(500).build();
71          }
72          return Response.status(200).entity(result.toString()).build();
73      }

```

- Dòng 24 định nghĩa path của các resource của ứng dụng <http://localhost:8080/rest/>
- Dòng 26 biểu thị method type của phương thức login là POST
- Dòng 27 biểu thị payload của data được truyền nhận dưới dạng JSON
- Dòng 28 biểu thị tên Resource của phương thức login là /login. Điều này có nghĩa là muốn sử dụng phương thức login thì từ ứng dụng khác cần sử dụng đường link: <http://localhost:8080/rest/login>

```

19  public class Main {
20      public static void main(String[] args) throws Exception {
21          ServletContextHandler context =
22              new ServletContextHandler(ServletContextHandler.SESSIONS);
23          context.setContextPath("/");
24          Server jettyServer = new Server(8080);
25          jettyServer.setHandler(context);
26          ServletHolder jerseyServlet =
27              context.addServlet(ServletContainer.class, "/*");
28          jerseyServlet.setInitOrder(0);
29          jerseyServlet.setInitParameter(
30              "jersey.config.server.providerclassnames",
31              MyAPIService.class
32                  .getCanonicalName()
33          );
34          try {
35              jettyServer.start();
36              jettyServer.join();
37          } catch (Exception ex) {
38              Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
39          } finally {
40              jettyServer.stop();
41          }
42      }

```

- o Dòng 24 khởi tạo một jetty server trên port 8080. vì vậy để truy cập vào các API resource chúng ta cần truy cập vào theo port 8080.
- Để thử dùng các tài nguyên API chúng ta sử dụng trình ứng dụng Postman như bên dưới theo URL sau:  
<http://localhost:8080/rest/login>

cửa hàng chrome trực tuyến

haitanitt@



Postman

[Chạy ứng dụng](#)Do [www.getpostman.com](http://www.getpostman.com) cung cấp★★★★★ 9.118 | [Tiện ích](#) | 3.805.387 người dùng

☑ Hoạt động khi không có mạng

The screenshot shows the Postman application interface. At the top, there are buttons for NEW, Runner, Import, and Builder, with 'Builder' being the active tab. To the right are 'Team Library', 'SYNC OFF', 'Sign In', and various notification icons. A message at the top states: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. Below this, there's a search bar labeled 'Filter' and tabs for 'History' (which is selected) and 'Collections'. On the left, under 'History', there's a section for 'Today' with a single entry: 'POST http://localhost:8080/rest/login'. The main workspace shows a POST request to 'http://localhost:8080/rest/login'. The 'Body' tab is selected, showing raw JSON data:

```

1 {"user": "activeStudy",
2 "password": "123456"
3 }

```

Below the body, the response status is 'Status: 200 OK' and 'Time: 36 ms'. The 'Headers' tab shows three entries: 'Content-Type: application/json', 'Content-Length: 18', and 'Date: Mon, 10 Dec 2018 09:45:11 GMT'. The 'Body' tab displays the response JSON:

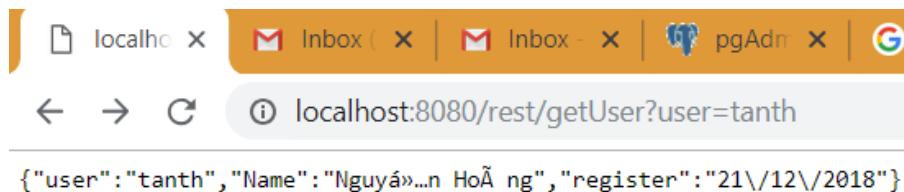
```

1 {
2   "code": "0",
3   "explain": "Success"
4 }

```

- Để truy vấn thông tin user thì mở trình duyệt và truy cập vào URL sau:

<http://localhost:8080/rest/getUser?user=tanth>



### 3.16 Bộ thu dọn rác (Garbage Collector) trong Java

- Nếu ai đã từng làm việc với C/C++ thi chắc cũng đã đều đau đầu và ám ảnh bởi Memory Leaks, một hiện tượng rò rỉ bộ nhớ do bộ nhớ được cấp phát ra bởi lệnh new mà không được thu hồi lại sau khi sử dụng xong. Đối với Java thì lập trình viên được giải phóng gần như hoàn toàn về việc này. Chính điều này làm nên sức mạnh của java, tạo ra cho chương trình viết bằng Java trở lên bền bỉ và mạnh mẽ.

- Đối với Java, khi quá trình hoạt động của garbage collector có nguy cơ ảnh hưởng tới performance của ứng dụng, lập trình viên có thể tùy chỉnh garbage collector theo nhiều cách khác nhau để đạt được hiệu năng mong muốn.
- Đối với các lập trình viên bình thường và viết các ứng dụng thông thường thì vấn đề Garbage Collector ít được để ý và sử dụng. Tuy nhiên để có được sản phẩm ứng dụng tốt hơn đòi hỏi lập trình viên cần phải có hiểu biết Garbage Collector để sử dụng cho hợp lý.
- Về cơ chế hoạt động của Garbage Collector thì có vài điểm như sau:
  - o Khi một đối tượng trên bộ nhớ Heap không được tham chiếu đến bởi bất kì một biến đối tượng nào nữa thì garbage collector sẽ quy hoạch và tiến hành dọn dẹp các bộ nhớ này vào một thời điểm thích hợp. Vì vậy để bộ nhớ được giải phóng tốt ta cần chú ý unreferenced các đối tượng vào thời điểm thích hợp để đối tượng được giải phóng bằng việc gán biến đối tượng bằng null;
 

```
myObjectVariable = null;
```
  - o Khi cần yêu cầu garbage collector hoạt động ta có thể chủ động gọi tới 2 một trong hai phương thức sau:
    - Cách 1:
 

```
18 | System.gc();
```
    - Cách 2:
 

```
19 | Runtime rt = Runtime.getRuntime();
20 | rt.gc();
```
  - o Khi đối tượng được garbage collector dọn dẹp thì phương thức finalize() của đối tượng sẽ được gọi trướing. Phương thức này chúng ta có thể override để thực hiện việc dọn dẹp tùy theo mục đích của chúng ta.
 

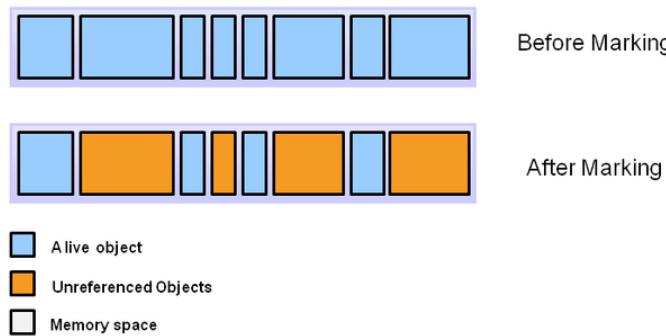
```
17 |     protected void finalize() throws Throwable {
18 |         super.finalize();
19 |         // clean up code follows.
}
```

Để kiểm soát bộ nhớ ta có thể dùng các lệnh sau để biết lúc nào cần gọi gc()

```
19 |     Runtime rt = Runtime.getRuntime();
20 |     rt.gc();
21 |     long freeMemory = rt.freeMemory();
22 |     long totalMemory = rt.totalMemory();
23 |     System.out.println("freeMemory : " + freeMemory);
24 |     System.out.println("totalMemory: " + totalMemory);
```

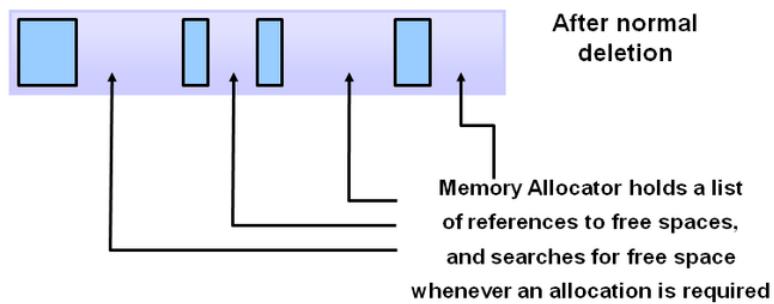
- Garbage Collector sẽ lên lịch và chạy vào các thời điểm thích hợp và thực hiện 3 bước cơ bản sau:

- Đánh dấu những đối tượng còn được sử dụng và những đối tượng không còn được sử dụng



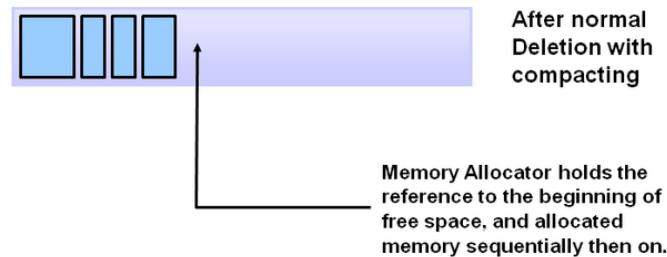
- Xóa bỏ các đối tượng không còn được sử dụng

### Normal Deletion



- Dồn các đối tượng vào để tránh phân mảnh bộ nhớ, làm tối ưu quá trình sử dụng bộ nhớ.

### Deletion with Compacting



- o Chúng ta hầu như không biết khi nào thì Garbage Collector thực hiện dọn dẹp bộ nhớ vào thời điểm nào, ngay cả khi chúng ta gọi tới System.gc() hay Runtime.gc(), ta vẫn không thể chắc chắn được garbage collector có chạy hay không.

### 3.17 Join dự án Maket Online.