

Tài liệu

Lập trình Kotlin toàn tập



Thông tin tác giả:

- ✓ Trần Duy Thành (Profile <https://duythanhcse.wordpress.com/contact/>)
- ✓ Blog chia sẻ kiến thức: <http://duythanhcse.wordpress.com/>
- ✓ Website khóa học trực tuyến: <https://communityuni.com/>
- ✓ Phone: 0987773061
- ✓ Email: duythanhcse@gmail.com

TPHCM-2017

Mục Lục

Lời giới thiệu	4
Bài 1: Có nên học Kotlin?	5
Bài 2-Cài đặt công cụ lập trình Kotlin	10
Bài 3-Tạo ứng dụng Kotlin đầu tiên.....	20
Bài 4-Cách xuất dữ liệu ra màn hình Kotlin	27
Bài 5 – Các cách ghi chú quan trọng khi lập trình Kotlin	29
Bài 6-Kiểu dữ liệu và cách khai báo biến trong Kotlin	32
Bài 7 – Ép kiểu dữ liệu trong Kotlin	35
Bài 8 – Các toán tử quan trọng trong Kotlin	38
Bài 9- Nhập dữ liệu từ bàn phím với Kotlin	47
Bài 10- Câu trúc điều khiển if else trong Kotlin.....	51
Bài 11-Biểu thức when trong Kotlin	56
Bài 12-Vòng lặp for trong Kotlin	62
Bài 13-Vòng lặp while trong Kotlin.....	69
Bài 14-Vòng lặp do while trong Kotlin	73
Bài 15-Xử lý biệt lệ trong Kotlin.....	78
Bài 16-Cách gỡ lỗi Kotlin bằng công cụ Debug.....	82
Bài 17-Các thư viện quan trọng thường dùng trong Kotlin	89
Bài 18- Xử lý chuỗi trong Kotlin.....	98
Bài 19- Xử lý mảng một chiều trong Kotlin	106
Bài 20- Xử lý mảng hai chiều trong Kotlin.....	111
Bài 21-Collections trong Kotlin.....	115
Bài 22-Lập trình hướng đối tượng trong Kotlin – phần 1	121
Bài 23-Lập trình hướng đối tượng trong Kotlin – phần 2	129
Bài 24-Lập trình hướng đối tượng trong Kotlin – phần 3	139
Bài 25-Lập trình hướng đối tượng trong Kotlin – phần 4	146
Bài 26-Lập trình hướng đối tượng trong Kotlin – phần 5	150
Bài 27-Alias và cơ chế gom rác tự động trong Kotlin-OOP phần 6	157
Bài 28-Extensions Method trong Kotlin-OOP phần 7	162
Bài 29-Xử lý Text File trong Kotlin.....	167

Bài 30-Xử lý Serialize File trong Kotlin	172
Bài 31-Xử lý XML File trong Kotlin	176
Bài 32-Xử lý JSON trong Kotlin – Bài 1	182
Bài 33-Xử lý JSON trong Kotlin – Bài 2	190
Bài 34-Đọc JSON tỉ giá hối đoái của Ngân Hàng Đông Á trong Kotlin – Bài 3	196
Bài 35-Thiết kế giao diện trong Kotlin – phần 1	202
Bài 36-Thiết kế giao diện trong Kotlin – phần 2	208
Bài 37-Thiết kế giao diện trong Kotlin – phần 3	214
Bài 38-Thiết kế giao diện trong Kotlin – phần 4	224
Bài 39-Thiết kế giao diện trong Kotlin – phần 5	255
Bài 40-Kết xuất Executable cho Kotlin [Kết thúc khóa học Kotlin]	263
Tài liệu tham khảo	271

Lời giới thiệu

Theo nhiều lời đề nghị của mọi người, Tui soạn thảo lại các bài lập trình Kotlin trên Blog <https://duythanhcse.wordpress.com/kotlin/kotlin-co-ban-den-nang-cao/> thành Ebook để giúp các bạn dễ học tại máy.

Kotlin và Java là song kiếm hợp bích, để học tốt Kotlin thì theo Tui các bạn nên học tốt Java trước. Hai ngôn ngữ này sẽ tương hỗ cho nhau trong quá trình viết mã lệnh. Google đã công Kotlin trở thành ngôn ngữ chính thống cho việc triển khai các dự án Android, do đó tương lai nó có tiềm năng rất lớn. Các bạn cố gắng học tốt Kotlin, Tui đã chủ ý biên soạn theo thứ tự từ thấp lên cao do đó các bạn nên học theo từng bài. Năm vũng Kotlin tốt sẽ tạo cơ hội trong tương lai cho các bạn, vì tương lai sẽ có làn sóng mạnh mẽ về tuyển dụng lập trình viên Android với Kotlin.

Trong quá trình biên soạn sẽ không tránh khỏi những sai sót, quý độc giả vui lòng hoan hỉ lượng thứ và gửi thư góp ý về cho: Trần Duy Thành (duythanhcse@gmail.com) để các phiên bản sau được chu đáo hơn.

Nếu quý độc giả có share và dùng làm tài liệu cho trung tâm, trường học... thì vui lòng ghi rõ nguồn gốc tài liệu này.

Xin chân thành cảm ơn

Trần Duy Thành.

Bài 1: Có nên học Kotlin?

Mấy ngày này cái tên Kotlin đã tạo nên một cơn địa chấn làm rung chuyển giới công nghệ, bạn đã xem phim ‘Đường Sơn Đại Địa Chấn’ chưa? nếu bộ phim vô cùng hay này đã cướp đi không biết bao nhiêu nước mắt của khán giả thì Kotlin làm điều ngược lại, nó lan tỏa không biết bao nhiêu nụ cười cho giới lập trình viên bởi nhiều tiện ích mà nó đem lại. Đặc biệt ngày 17/05/2017 vừa rồi Google đã công bố Kotlin trở thành ngôn ngữ lập trình Android chính thống giáo, từ phiên bản Android Studio 3.0 các lập trình viên có thể tha hồ tung hoành!

Và Tui dự đoán rằng: Trong tương lai sẽ có làn sóng mạnh mẽ về tuyên dụng lập trình viên Android bằng ngôn ngữ Kotlin, các công ty sẽ rất khát nhân lực, các bạn cần nhanh chóng nghiên cứu Kotlin để đi đầu về công nghệ.

Nếu bạn còn bảo lưu quan điểm Chậm Mà Chắc, thì Tui nghĩ nó không còn đúng nữa. Thời đại này khác xưa rồi, các bạn phải Nhanh Mà Chắc mới hơn người ta được, đừng chờ cho tới khi Kotlin quá phổ biến thì lúc đó bạn là người đến sau. Hãy chiến đấu ngay từ bây giờ để đi đầu về công nghệ!

Hi hi hi, nghe tới đây bạn **Dã Ghiền Kotlin** chưa? Ngày xưa Tui học Văn là dốt nhất lớp, toàn bị 4.5 điểm, nên cố gắng lắm mới viết được một chút ít giới thiệu về Kotlin



ha ha – nhìn hình này có vẻ Toptal nói Java già cỗi

Kotlin có nhiều ưu điểm, ở đây Tui liệt kê một số để các bạn tham khảo (dĩ nhiên các bạn có thể tìm hiểu thêm):



Ngắn gọn như thế nào?

- Ta có thể dễ dàng viết các POJO (**Plain Old Java Object**) trên một dòng :

```
data class Customer(val name: String, val email: String,  
val company: String)
```

- Ta có thể dùng Lambda để lọc dữ liệu một cách nhanh chóng:

```
val positiveNumbers = list.filter { it > 0 }
```

- Ta có thể tạo đối tượng bằng Singleton:

```
object ThisIsASingleton {  
    val companyName: String = "https://ssoftinc.com/"  
}
```

Và còn nhiều cách viết ngắn gọn khác nữa, các bạn có thể tham khảo thêm trên <http://kotlinlang.org/>

An toàn như thế nào?

Kotlin tự động kiểm tra lỗi biến dịch Null pointer exception, các hành vi trên tập dữ liệu null, tự động ép kiểu đúng một cách chính xác cho ta, ví dụ so sánh:

KOTLIN	JAVA
<code>var str1: String? = null str1?.trim() // doesn't run</code>	<code>String str1 = null; str1.trim(); // runs and crashes</code>
<code>str1 = "Not null anymore" str1?.trim() // does run</code>	<code>str1 = "Not null anymore"; str1.trim(); // runs</code>
<code>str1!!.trim() // runs anyway</code>	
<code>val str2: String = "I am not null" str2.trim() // no need for "?"</code>	<code>String str2 = "I am not null"; str2.trim(); // runs</code>

Đa năng như thế nào?

Phải nói Kotlin có thể làm các multiplatform applications. Có thể build Kotlin cho Server-side , cho Android, cho Javascript, Native....

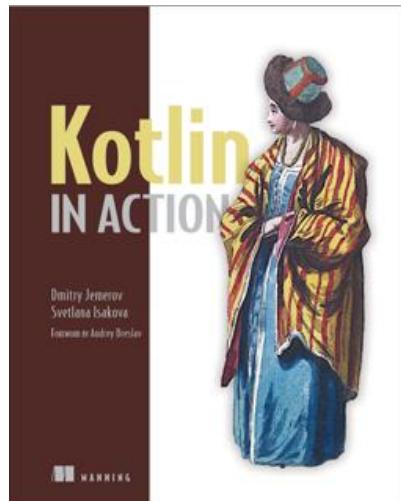
Khả năng tương tác như thế nào?

Kotlin có thể sử dụng được 100% các thư viện từ JVM, có thể dễ dàng từ Kotlin triệu gọi Java và từ Java triệu gọi Kotlin. Giúp các Lập trình viên không lo lắng về việc chuyển đổi coding, tăng khả năng tương tác mạnh mẽ trong hệ thống.

Ngoài ra Kotlin còn có thể dễ dàng lập trình trên nhiều công cụ khác nhau: Website, Eclipse, Netbeans, Android Studio, JetBrains... Tài liệu lập trình phong phú, cộng đồng hỗ trợ Kotlin ngày càng không ngừng phát triển.

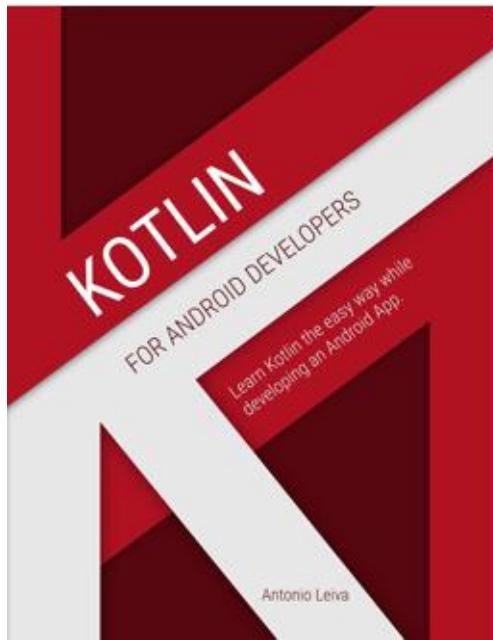
Các cuốn sách Lập trình viên có thể nghiên cứu:

1.Kotlin in Action



Cuốn sách có 11 chương, giúp bạn hiểu rõ về Kotlin từ cơ bản tới nâng

2.Kotlin for Android Developers



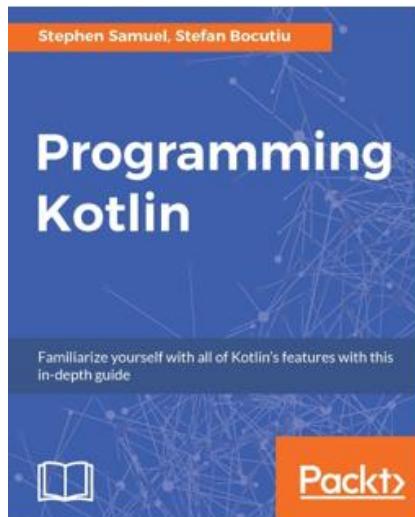
Sách dành cho những ai đã rành về Kotlin, tiếp tục phát triển Kotlin bên Android (phần đầu vẫn dạy về Kotlin), được xé nhỏ thành 26 chương giúp ta dễ dàng học

3.Modern Web Development with Kotlin



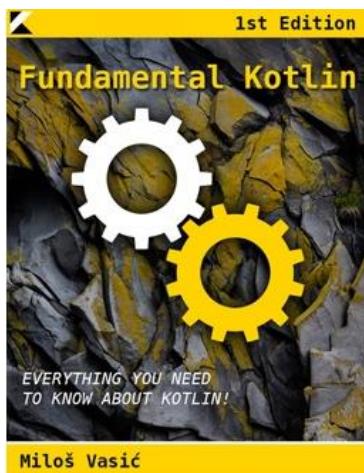
Cuốn sách dạy về Web với Kotlin, đặc biệt EcmaScript 6 chuẩn mới nhất, Json....Các bạn quan tâm có thể học, khoảng 115 trang.

4.Programming Kotlin



Cuốn này cũng tương tự, giúp ta có thể học tốt Kotlin. Bố trí thành 13 chương (420 pages) các bạn có thể bám theo cuốn này để học

5.Fundamental Kotlin



Cuốn sách này khá hay, bạn có thể tham khảo.

Chúc các bạn nhanh chóng học tốt Kotlin, hẹn gặp các bạn ở những bài sau

Trần Duy Thành (<https://ssoftinc.com/>)

Bài 2-Cài đặt công cụ lập trình Kotlin

Ở [bài 1](#) Tui đã trình bày lý do vì sao nên học Kotlin, Trong bài này Tui sẽ hướng dẫn các bạn cách cài đặt công cụ lập trình Kotlin.

Để lập trình được Kotlin các bạn có thể sử dụng Website để thử nghiệm online <https://try.kotlinlang.org/>

Hoặc cài đặt phần mềm IntelliJ IDEA , Eclipse Neon , Command Line Compiler , Build Tools (Ant, Maven, Gradle, Griffon (external support))

Bài này Tui sẽ hướng dẫn cách cài đặt phần mềm IntelliJ IDEA để lập trình Kotlin (vì xuyên suốt các bài hướng dẫn lập trình Kotlin thì Tui sẽ dùng công cụ này để minh họa)

Trước tiên bạn cần cài JDK vào máy trước (Kotlin chạy trên JVM, cài bản 1.8 trở lên), khóa học Kotlin thường dành cho những ai đã rành về Java. Bước này các bạn tự xử nhé.

Có 2 trường hợp để tải phần mềm IntelliJ IDEA:

1.Nếu bạn là lập trình viên bình thường

2.Nếu bạn là Teacher hoặc Student (dành cho Education)

Bây giờ Tui sẽ hướng dẫn chi tiết 2 trường hợp tải phần mềm này

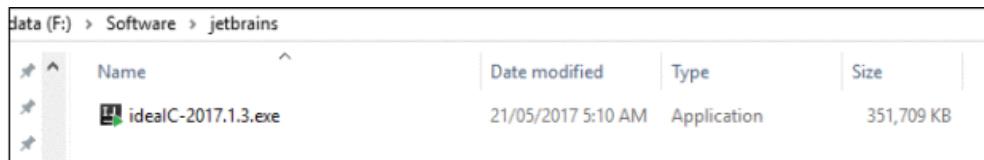
Trường hợp 1: Nếu bạn là lập trình viên bình thường

Các bạn tải bản Community của IntelliJ IDEA tại link sau:

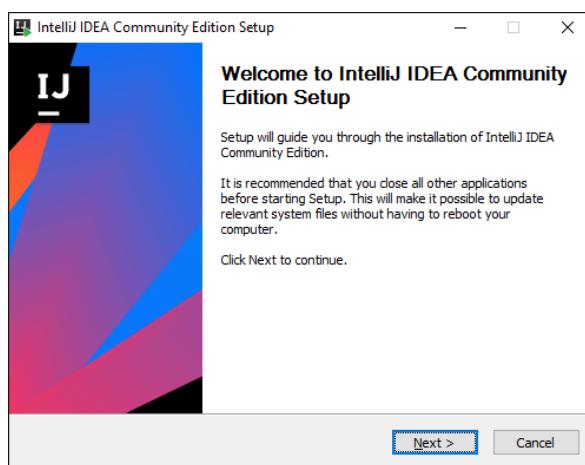
<http://www.jetbrains.com/idea/download/index.html>



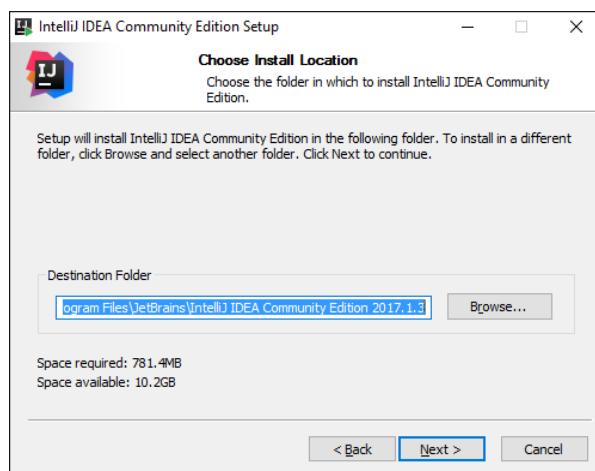
Ở màn hình trên, ta chọn Download Exe trong mục Community, tại thời điểm Tui viết bài hướng dẫn này là ngày 21/05/2017 nên bạn sẽ có kết quả sau (tùy thuộc vào thời điểm bạn tải khác nhau mà có thể có version khác):



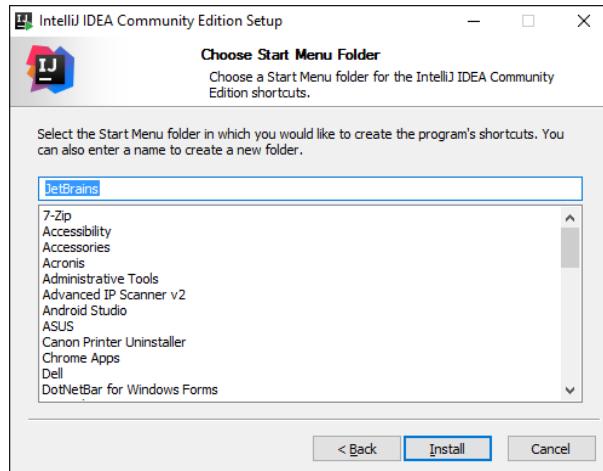
Ta thấy với phiên bản hiện tại thì có tập tin “ideaIC-2017.1.3.exe”, dung lượng hơn 351MB. Để cài đặt ta double click vào tập tin vừa tải về máy:



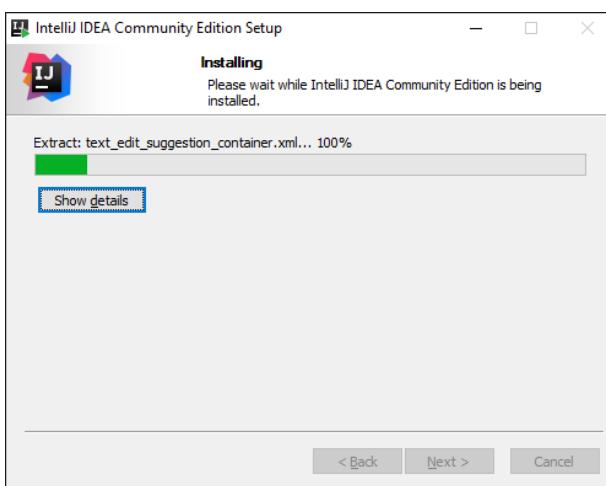
Bấm Next để tiếp tục, màn hình yêu cầu chọn nơi cài đặt sẽ hiển thị ra như dưới đây:



Ta có thể để mặc định rồi bấm Next, Chương trình sẽ hiển thị các cấu hình lựa chọn trong quá trình cài đặt, Ta chọn cấu hình như trên rồi bấm Next -> màn hình yêu cầu chọn Start Menu xuất hiện:



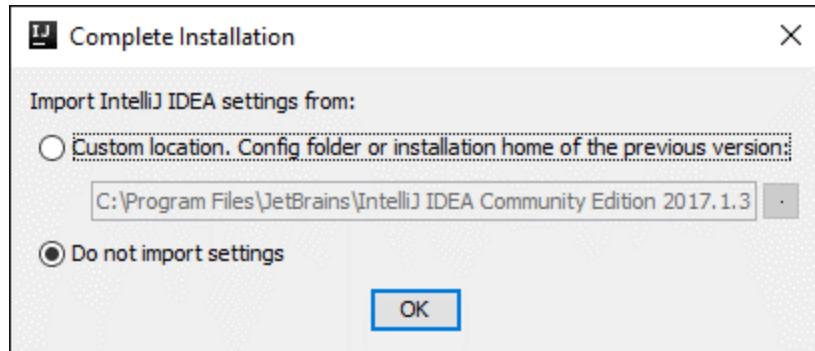
Ta để mặc định rồi bấm Install, chờ chương trình hoàn tất việc cài đặt:



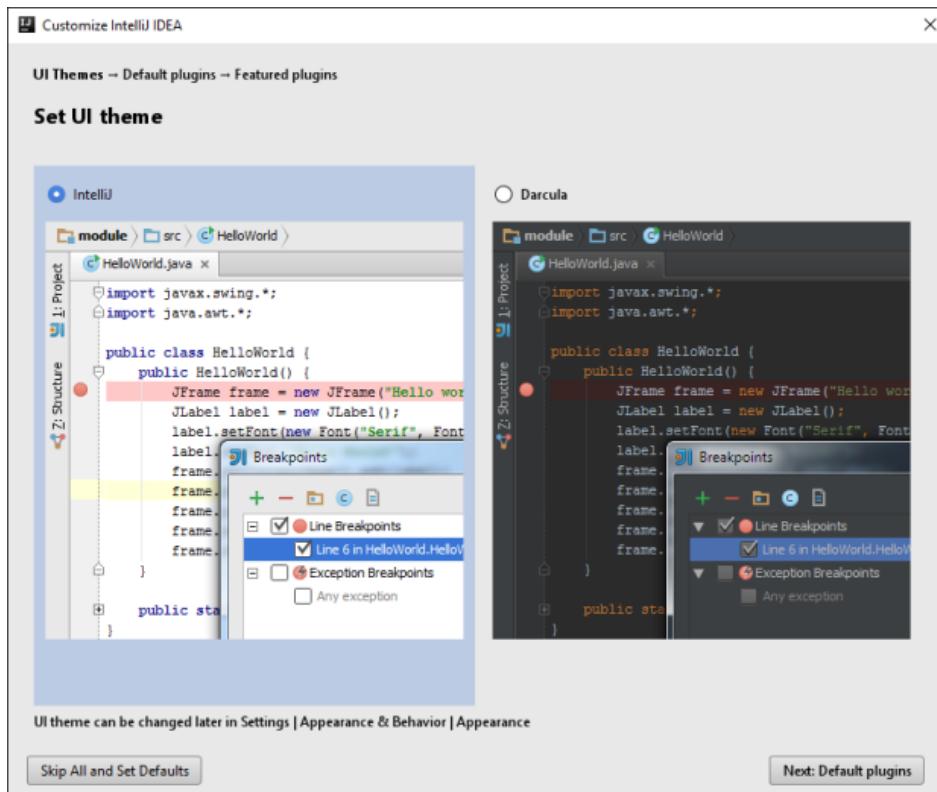
Sau khi cài đặt thành công, ta có giao diện thông báo như dưới đây:



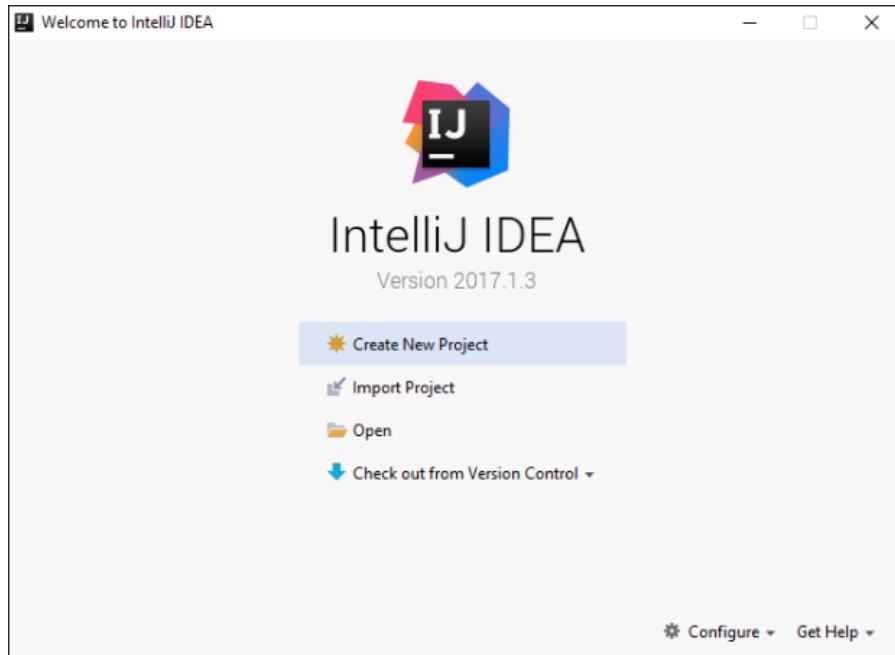
Bấm Finish để hoàn tất việc cài đặt, nếu muốn hoàn tất và khởi động luôn phần mềm thì checked vào “Run IntelliJ IDEA Community Edition”, ta cũng có thể quan sát ngoài màn hình Desktop đã có shortcut để chạy phần mềm. Nếu là lần đầu chạy phần mềm bạn sẽ gặp cửa sổ sau:



Ta chọn Do not import settings rồi bấm OK, màn hình yêu cầu thiết lập Theme cho công cụ xuất hiện:



Có 2 màn hình nền mặc định ở trên, tùy bạn lựa chọn, sau đó bấm Skip All And Set Defaults cho lẹ, dưới đây là màn hình sau khi đã cấu hình xong IntelliJ IDEA, các lần sau khởi động sẽ tương tự:



Trường hợp 2: Nếu bạn là Teacher hoặc Student (dành cho Education)

Vào link : <https://www.jetbrains.com/student/>

Are you learning Java, PHP, Ruby, Python, JavaScript, Objective-C or .NET technologies?

Or maybe you just plan to? Do it right from the start, with award-winning professional developer tools from JetBrains. And the best part: it's free of charge.

All Products Pack

Get access to all desktop products including IntelliJ IDEA Ultimate; ReSharper Ultimate and other IDEs. All you need to apply is to be a student and have access to your student email address or a valid ISIC card.

APPLY NOW

Ta chọn Apply Now, màn hình đăng ký sẽ xuất hiện như dưới đây:

JetBrains s.r.o. (CZ) | https://www.jetbrains.com/shop/eform/students

1st Visited | Android | Khóa mới | Welcome | University ... | GiaoTrinh - Google Dr... | Website Chăm Sóc Kh... | 101 LINQ Samples in C# | Thành Thạo LINQ tron...

JetBrains Products for Learning

Apply with:

- UNIVERSITY EMAIL ADDRESS
- ISIC/ITIC MEMBERSHIP
- OFFICIAL DOCUMENT

Status:

- I'm a student
- I'm a teacher

Name:

Thanh	Tran
-------	------

Our software will be registered to your real name.

Email address:

Your valid university email address, e.g. john.smith@mit.edu. We'll send you further instructions.

APPLY FOR FREE PRODUCTS

Ở màn hình trên bạn chọn Universities Email Address. Nếu là Giảng Viên thì chọn I'm a Teacher, còn nếu là Sinh Viên thì chọn I'm a Student

Nộp đầy đủ tên và email rồi nhấn Apply For Free Products

Khi bạn nhấn nút này thì sẽ có 1 Email gửi tới email Education của bạn để yêu cầu bạn xác thực, nội dung giống như sau:

UEL

Mail

Compose

Inbox (527)

Starred

Sent Mail

Drafts (3)

Việc 2016

Viết Đề tài cấp cơ ...

Viết Đề tài NCKH SV

Search people...

Hi,

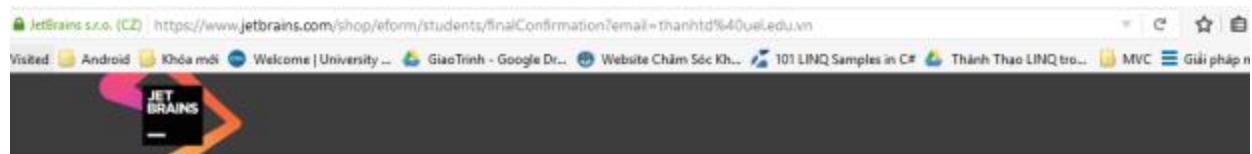
You've received this email because your email address was used for registering/updating a JetBrains Educational Pack.

Please follow this link to confirm your intention:

[Confirm Request](#)

Yours truly,
JetBrains Team
<https://www.jetbrains.com>
The Drive to Develop

Ở màn hình Email xác thực, bạn nhấp vào nút **Confirm Request** để xác thực. Khi nhấp vào nút này bạn sẽ thấy một màn hình thông báo xác thực thành công như dưới đây:



JetBrains Products for Learning

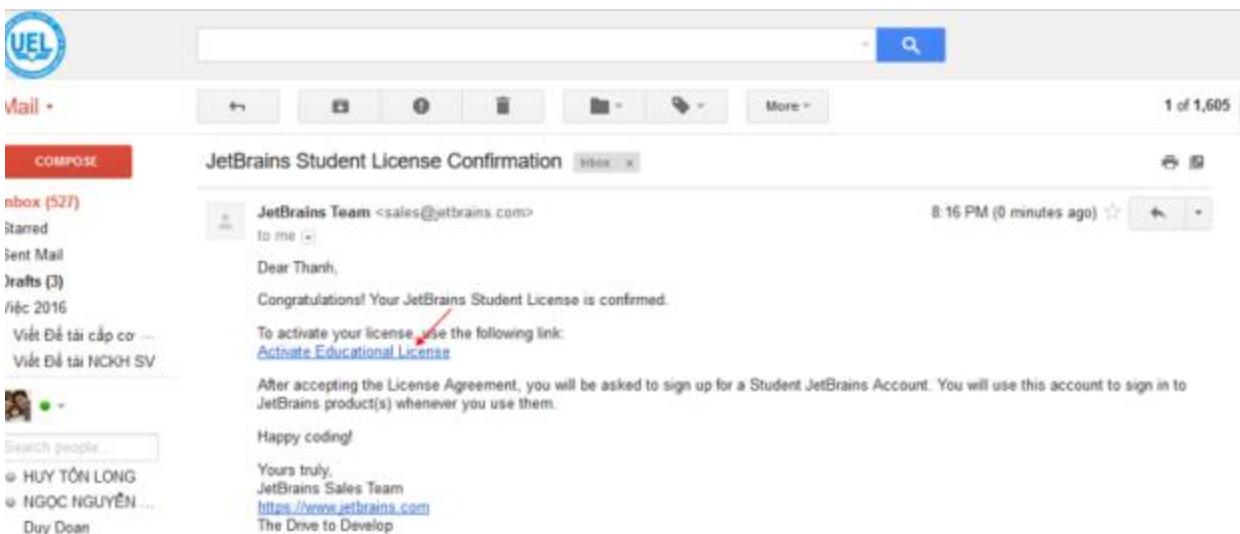
Congrats! You've been approved!

You are now entitled to use JetBrains products for free. Start learning by using Pro tools right now! To get started, please follow the instructions we have just emailed you at thanhtd@uel.edu.vn.

For more details you can also check out the [JetBrains Student program description page](#).

[« Discover our products](#)

Lúc này sẽ có 1 Email thứ 2 thông báo hướng dẫn cách Kích hoạt tài khoản education, bạn check email sẽ có nội dung tương tự dưới đây:



Bạn nhấp vào **Activate Educational License** để kích hoạt, lúc này sẽ có một Website xuất hiện, yêu cầu ta bấm Xác nhận:

TOOLBOX SUBSCRIPTION LICENSE AGREEMENT FOR EDUCATION

Version 2, Effective as of September 9th, 2016

IMPORTANT! READ CAREFULLY:

THIS IS A LEGAL AGREEMENT. BY CLICKING ON THE "I AGREE" (OR SIMILAR) BUTTON THAT IS PRESENTED TO YOU AT THE TIME OF YOUR PURCHASE, OR BY DOWNLOADING, INSTALLING, COPYING, SAVING ON YOUR COMPUTER, OR OTHERWISE USING JETBRAINS SOFTWARE, SERVICES OR PRODUCTS, YOU ARE BECOMING A PARTY TO THIS AGREEMENT, AND YOU ARE CONSENTING TO BE BOUND BY ALL THE TERMS AND CONDITIONS SET FORTH BELOW.

1. PARTIES

1.1. "JetBrains" or "We" means JetBrains s.r.o., having its principal place of business at Na hřebenech II 1718/10, Prague, 14700, Czech Republic, registered with Commercial Register kept by the Municipal Court of Prague, Section C, file 86211, ID.Nr.: 265 02 275.

1.2. "Licensee" or "you" means a student or an instructor specified in the Subscription Confirmation. For the purpose of this Agreement, a student is an individual who is enrolled at a recognized tertiary educational institution (university or college) that grants degrees requiring not less than the equivalent of two years of full-time study, and upon request by Licensor is able to provide proof of such enrollment. For the purpose of this Agreement, an instructor is an individual who conducts lectures and/or seminars at a recognized tertiary educational institution (university or college), and upon request by Licensor is able to provide proof of such involvement.

2. DEFINITIONS

2.1. "Agreement" means this Agreement.

Sau khi bấm Accept, bạn được yêu cầu nhập thông tin đăng nhập hệ thống:

Welcome to JetBrains Account!

Please complete the registration form below.

Email: thanhtd@uel.edu.vn

First Name: Thanh

Last Name: Tran

User Name: duythanhcsse

Please make sure you choose a strong password as your account will have access to your purchases

Password: *****

Repeat Password: *****

I have read and agree to [JetBrains Privacy Policy](#)

Submit

Nhập thông tin xong bạn nhấn Submit để đăng ký tài khoản, lúc này màn hình quản lý Phần mềm bản quyền sẽ xuất hiện như dưới đây:

The screenshot shows a browser window with the JetBrains account URL: https://account.jetbrains.com/licenses/assets. The page displays a success message: "1 × JetBrains Product Pack for Students Personal license were added." Below this, a section titled "1 License" is shown. A red arrow points to the "Download" button under the heading "JetBrains Product Pack for Students". To the right, the License ID is listed as "9JNB87X54E". The page details the license information: Licensed to "Thanh Tran", License restriction "For educational use only", and Valid through "May 28, 2018". It also lists the included products: IntelliJ IDEA Ultimate, dotCover, RubyMine, ReShaper, AppCode, WebStorm, ReSharper C++, CLion, PhpStorm, dotTrace, dotMemory, and PyCharm. At the bottom, there is a note: "After downloading and installing the software, simply run it and follow the on-screen prompts to sign in with your JetBrains Account."

Bạn nhấp vào Download, nó xô ra nhiều phần mềm. Cần phần mềm bản quyền nào thì cứ nhấp chọn mà tải:

The screenshot shows the "JetBrains Product Pack for Students" download page. A red arrow points to the "IntelliJ IDEA Ultimate" row in the list of available downloads. The table lists the following products and their download links:

App	Version	Download
AppCode	2017.1	Download
CLion	2017.1	Download
DataGrip	2017.1	Download
IntelliJ IDEA Ultimate	2017.1	Download
PhpStorm	2017.1	Download
PyCharm	2017.1	Download
ReSharper	2017.1	Download
ReSharper C++	2017.1	Download
RubyMine	2017.1	Download
WebStorm	2017.1	Download
dotCover	2017.1	Download
dotMemory	2017.1	Download

To the right of the table, a sidebar lists additional products: ReShaper, AppCode, CLion, and DataGrip. Below the table, there is a note: "After downloading and installing the software, simply run it and follow the on-screen prompts to sign in with your JetBrains Account". At the bottom, there is a note: "Sync your past purchases to your JetBrains Account".

Ở trên ta chọn IntelliJ IDEA Ultimate để tải:



Download IntelliJ IDEA Ultimate 2017.1.3

WINDOWS

MAC

LINUX

Product: [IntelliJ IDEA Ultimate](#)

Version: 2017.1.3

Build: 171.4424.56

Released: May 16, 2017

[DOWNLOAD](#)

[DOWNLOAD ZIP](#)

498.56 MB

649.39 MB

[SHA256 checksum](#)

[SHA256 checksum](#)

Bạn bấm Download và tải, sau đó cài đặt giống trường hợp 1 nhé.

Như vậy tới đây Tui đã hướng dẫn xong cách tải và cài đặt công cụ lập trình Kotlin, bạn nào từng làm Android Studio thì thấy giao diện rất tương đồng đúng không?

Bài kế tiếp Tui sẽ hướng dẫn cách tạo 1 Project HelloWorld Kotlin, để có cảm giác lập trình với ngôn ngữ mới coóng này nhé.

Các bạn chú ý theo dõi

Chúc các bạn thành công

Trần Duy Thanh (<https://ssoftinc.com/>)

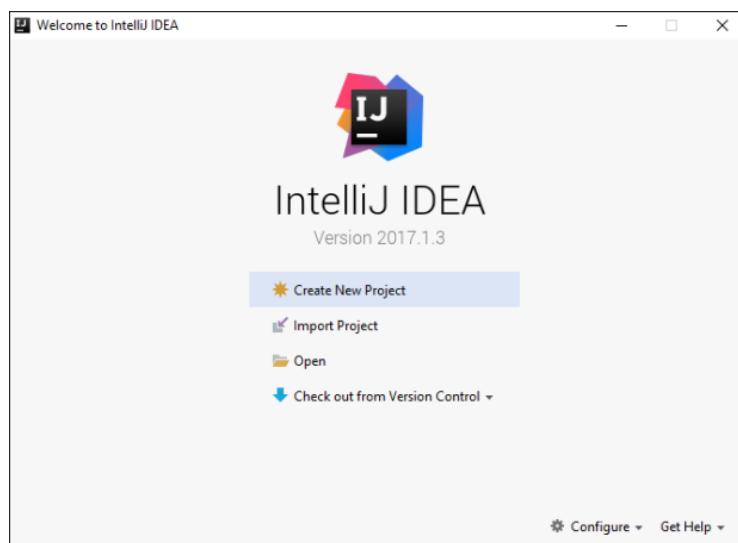
Bài 3-Tạo ứng dụng Kotlin đầu tiên

Ở [bài 2](#) chúng ta đã biết cách tải và cài đặt công cụ lập trình Kotlin. Trước khi đi vào chi tiết về Kotlin thì ta cần biết làm thế nào để tạo một Project Kotlin đầu tiên, ta thường nói Tiếng Anh cho sang miệng đó là “Hello World Project”. Ta không nói Tiếng Anh thì mọi người tưởng chúng ta dốt, nhưng đã nói rồi thì ... họ không còn nghi ngờ gì nữa.

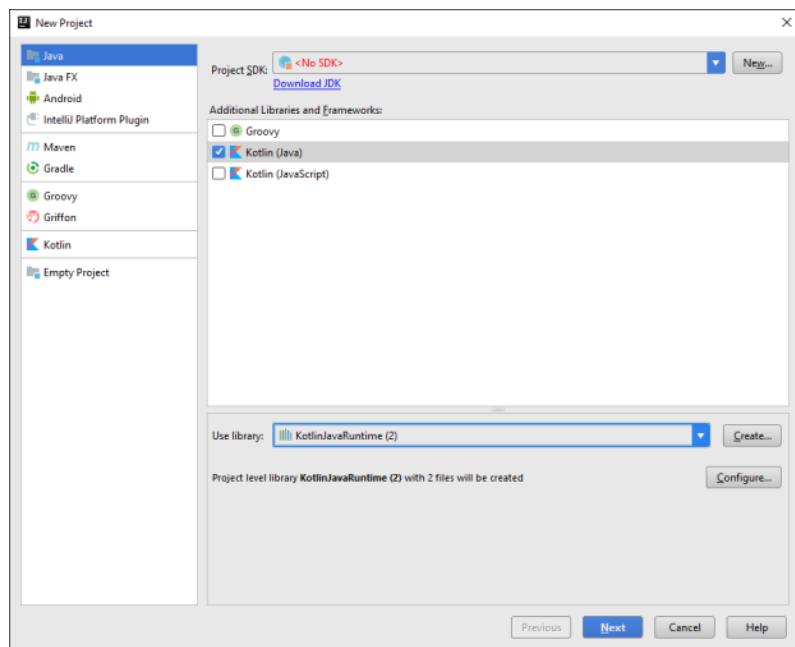
Khởi động IntelliJ IDEA, từ short cut ở màn hình Desktop ta double click để khởi động:



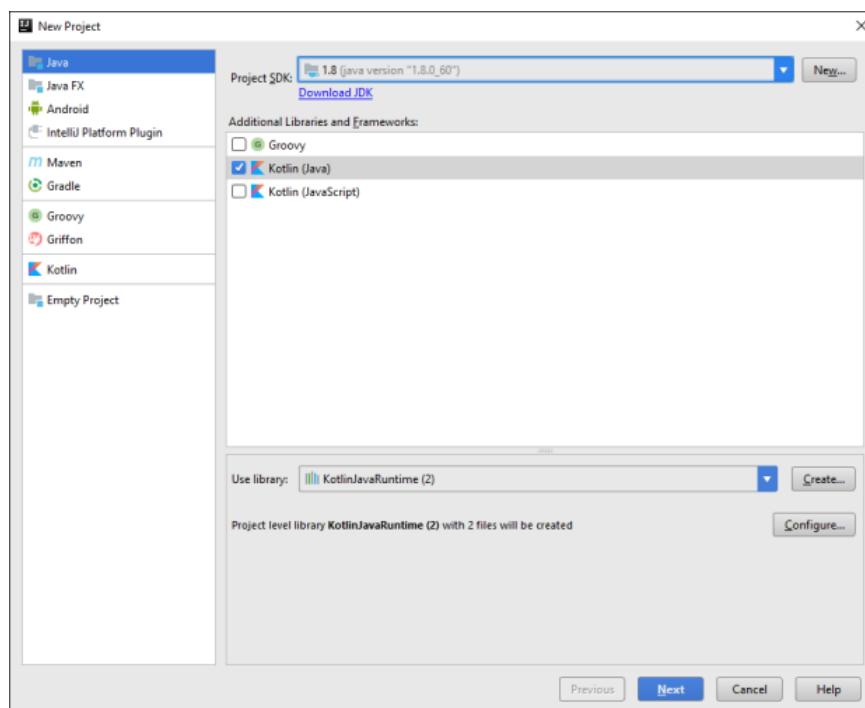
Màn hình Welcome của IntelliJ IDEA, ta bấm Create New Project:



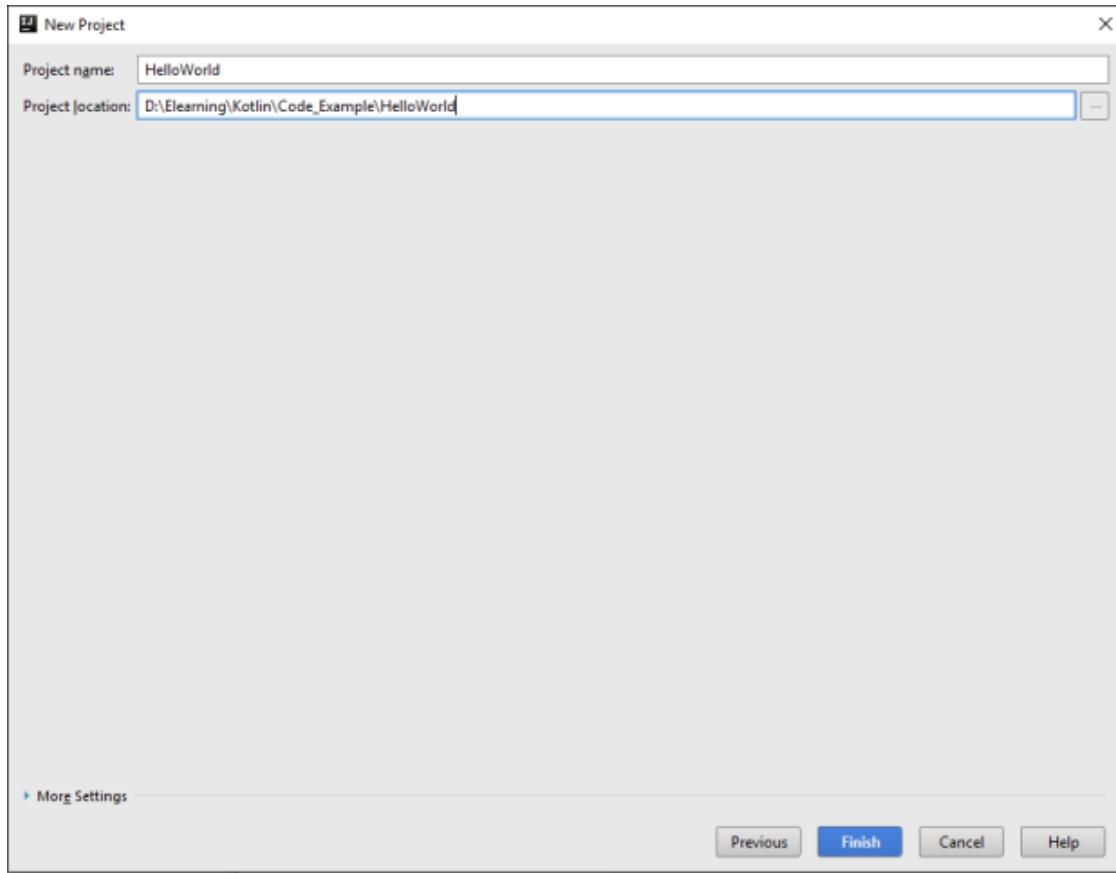
Sau khi bấm Create New Project, màn New Project xuất hiện:



Ở màn hình New Project bên trên, bạn chú ý góc phải trên cùng có button “New” cùng hàng với Project SDK. Đây chính là nơi chọn đường dẫn mà bạn đã cài đặt JDK, bạn bấm vào Button này để trỏ chính xác tới nơi mà bạn đã cài đặt (nên cài JDK từ bản 1.8 trở lên). Mục danh sách bên dưới các bạn checked vào Kotlin (Java). Sau khi cấu hình xong bạn sẽ có giao diện tương tự như dưới đây:



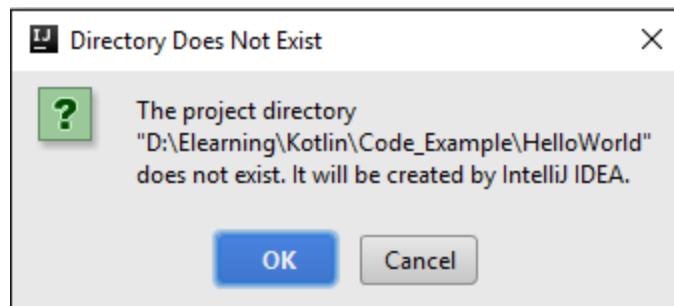
Bạn thấy đó, ở trên JDK đã được update, tiếp theo bạn bấm Next :



Mục Project name: Tên của dự án, bạn đặt “**HelloWorld**”

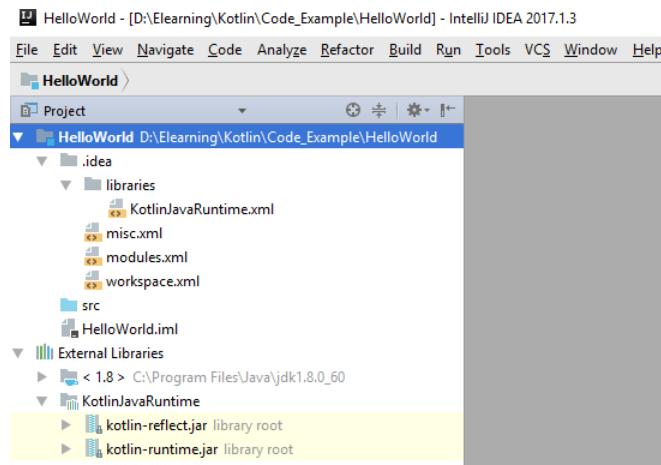
Mục Project Location: Nơi lưu trữ dự án, bạn trả lời thư mục mà bạn muốn lưu trữ.

Sau đó bấm Finish để tạo Project **HelloWorld**. Nếu chương trình kiểm tra thấy đường dẫn chưa tồn tại thì sẽ xuất hiện cửa sổ xác nhận để tạo:



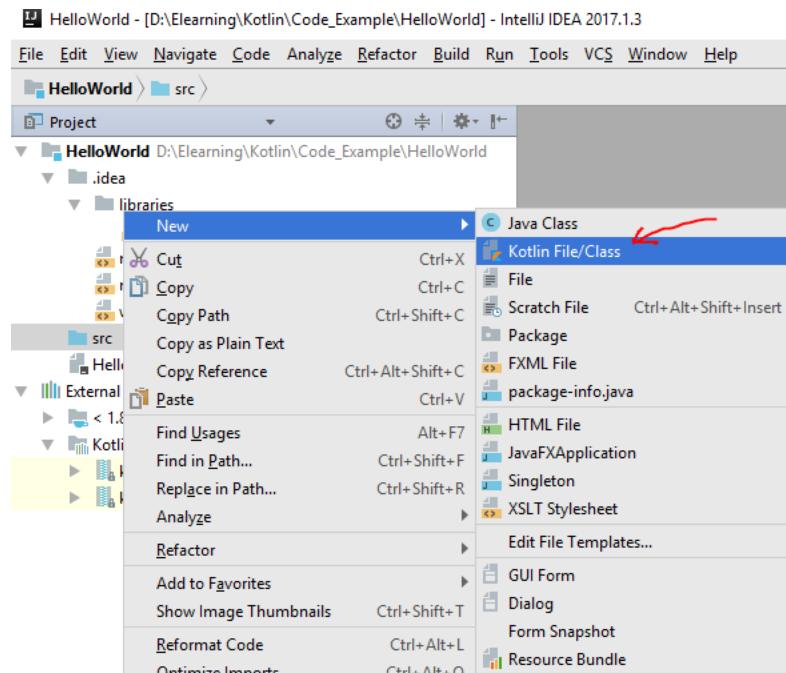
Ta bấm OK để đồng ý tạo đường dẫn lưu Project **HelloWorld**.

Đây là màn hình cấu trúc Project Kotlin được tạo ra:

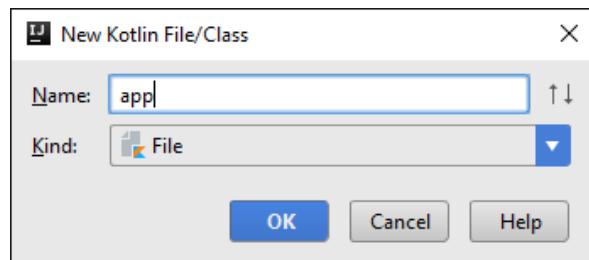


- Thư mục .idea cho ta các tập tin cấu hình, tham chiếu thư viện.
- Thư mục src là nơi lưu trữ các tập tin, lớp source code cho dự án.
- File HelloWorld.iml bản chất là một file XML, được lưu các thông số cấu hình mặc định cho dự án.
- External Libraries: Thư viện liên kết ngoài: Bắt buộc phải có JDK, KotlinJavaRuntime, các thư viện này sẽ được tham chiếu trong tập tin KotlinJavaRuntime.xml.

Để tạo một Mã nguồn bằng Kotlin ta tiến hành: Bấm chuột phải vào thư mục src/ chọn New/ chọn Kotlin File/Class:



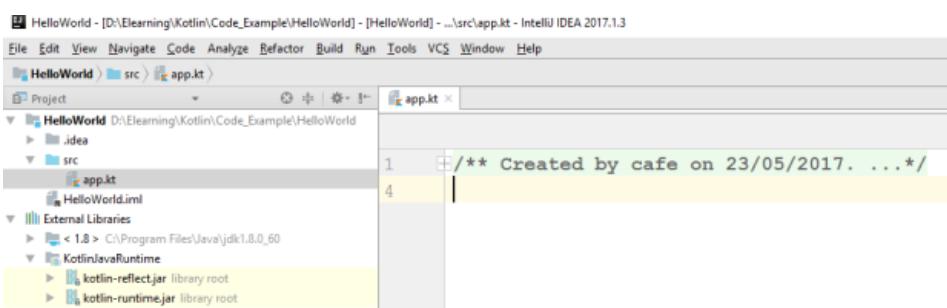
Màn hình yêu cầu tạo Kotlin File xuất hiện như dưới đây:



Mục Name: Bạn đặt tên tùy ý, ví dụ Tui đặt là app

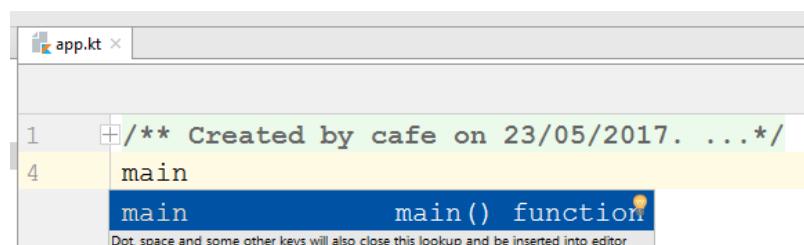
Mục Kind: Chọn File(bài này sẽ chọn File, các bài sau tùy trường hợp mà ta chọn các loại khác trong combobox)

Nhấn OK để tạo, ta thấy cấu trúc source code sẽ như sau:

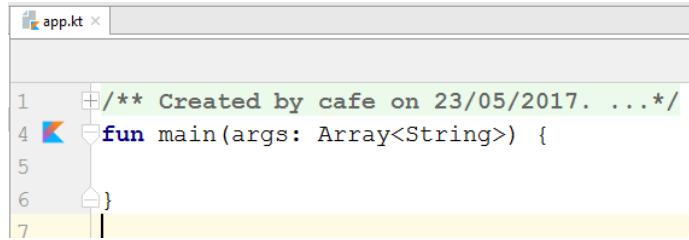


Như vậy bạn quan sát thấy, phần mở rộng của Kotlin là kt, ta tiến hành Coding để xuất ra dòng thông báo chất nhất quả đât ‘Hello World! I’m <http://ssoftinc.com/>’:

Trong màn hình soạn thảo coding của **app.kt**, bạn chỉ cần gõ chữ **main** rồi nhấn tổ hợp phím **ctrl+space**, hàm main đầy đủ sẽ được xuất hiện:

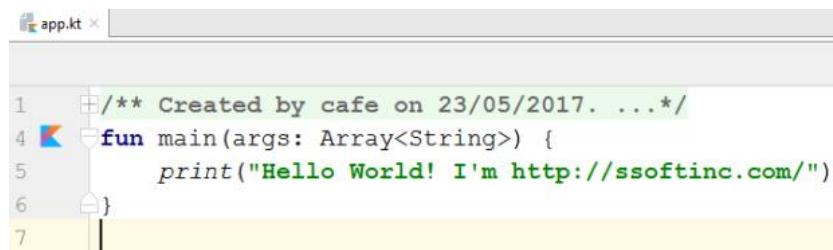


Khi bạn nhấn **Ctrl+space** bạn thấy dòng màu xanh bên trên xuất hiện với chữ **main()** function==>bạn chỉ cần nhấn **Enter** là tự động xuất hiện lệnh đầy đủ (mấy cái này gọi là Template, chả có gì cao siêu đâu, ta có thể tự cấu hình được. Còn đây là các Template mặc định của IntelliJ IDEA):



```
1  /** Created by cafe on 23/05/2017. ...*/
4  fun main(args: Array<String>) {
5
6  }
7
```

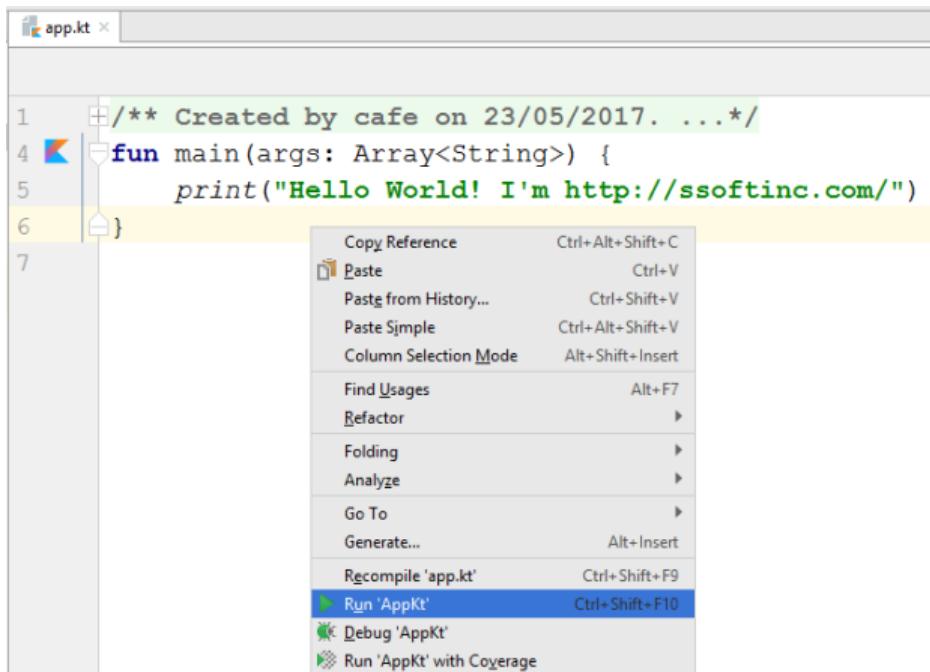
Ở trên bạn thấy cấu trúc hàm main, với từ khóa fun (tức là function), bên trong là các arguments input đầu vào khi chạy mã lệnh (thường được dùng để truyền thông số gọi qua lại giữa các ứng dụng khác nhau). Bạn muốn xuất dòng lệnh thông báo ra màn hình thì viết bên trong hàm main, ví dụ:



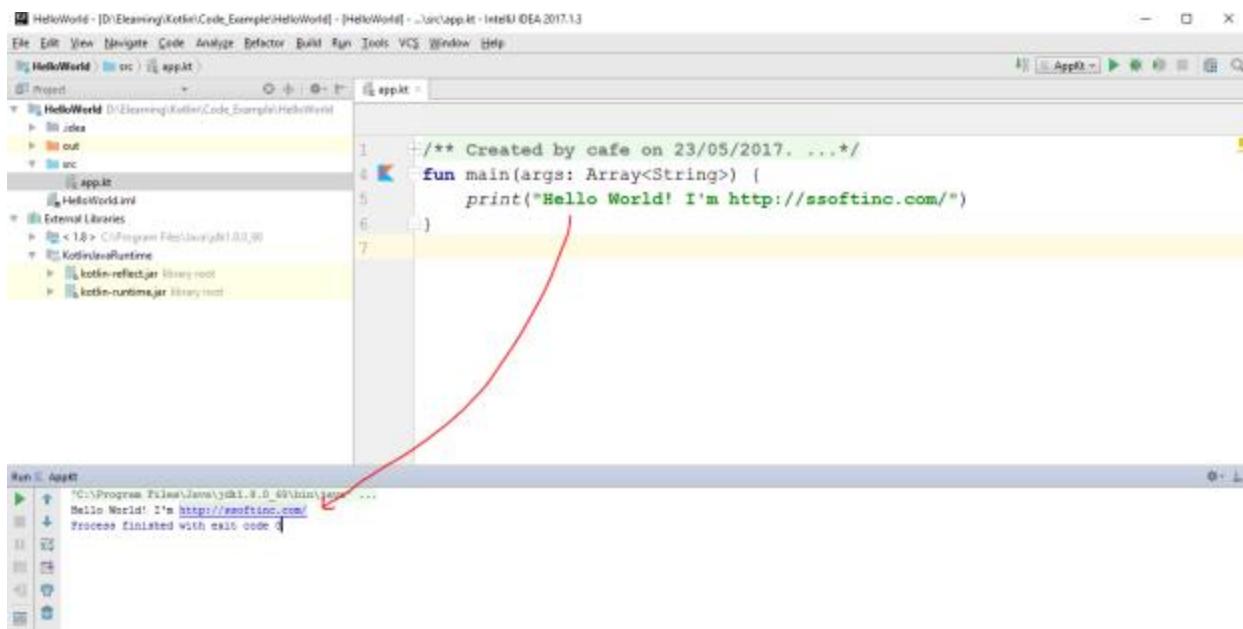
```
1  /** Created by cafe on 23/05/2017. ...*/
4  fun main(args: Array<String>) {
5      print("Hello World! I'm http://ssoftinc.com/")
6  }
7
```

Bạn quan sát nó có gì lạ với Java? kết thúc câu lệnh không phải gõ chấm phẩy đúng không?

Bây giờ làm sao để chạy được đoạn lệnh này? ta có thể vào menu Run/Run. Hoặc bấm chuột phải vào app.kt rồi chọn Run App.kt như hình dưới đây:



Bạn chờ chương trình biên dịch và chạy ra kết quả như dưới đây:



```
/* Created by cafe on 23/05/2017. ...*/
fun main(args: Array<String>) {
    print("Hello World! I'm http://ssoftinc.com/")
}
```

Run > AppKt

```
C:\Program Files\Java\jdk1.8.0_45\bin>java -jar C:\Users\cafe\IdeaProjects>HelloWorld\app.jar
Hello World! I'm http://ssoftinc.com/
Process finished with exit code 0
```

Như vậy Tui đã hướng dẫn xong chi tiết cách tạo một Project Kotlin ban đầu như thế nào, cũng như cách chạy nó, Các bạn làm bài này để thành thạo các thao tác cơ bản đầu tiên trước nhé.

Tải source code tại đây:

<http://www.mediafire.com/file/jccf8ghwar4d7de/HelloWorld.rar>

Chúc các bạn thành công

hẹn gặp lại các bạn ở các bài hướng dẫn tiếp theo

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 4-Cách xuất dữ liệu ra màn hình Kotlin

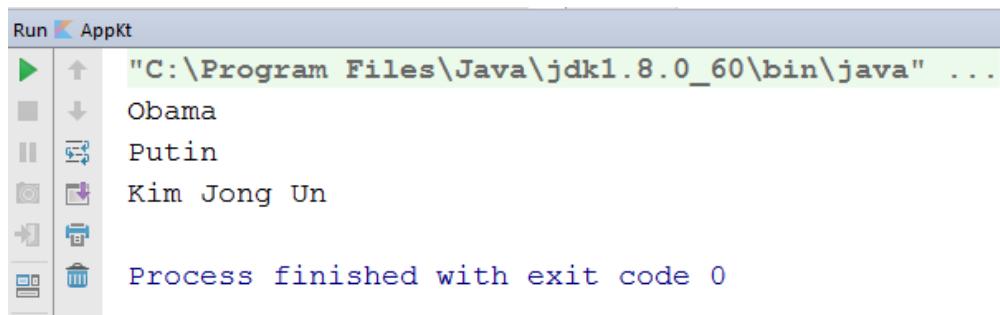
Như vậy ở [bài 3](#) chúng ta đã biết tạo một Project Kotlin như thế nào rồi, trong bài này chúng ta sẽ học cách thức xuất dữ liệu ra màn hình Kotlin.

Trong Kotlin để xuất dữ liệu ra màn hình ta dùng 2 hàm chính đó là print() và println(). Hai hàm này thuộc thư viện [kotlin.io](#)

hàm println() dùng để xuất dữ liệu trên các dòng khác nhau, ví dụ:

```
println("Obama")
println("Putin")
println("Kim Jong Un")
```

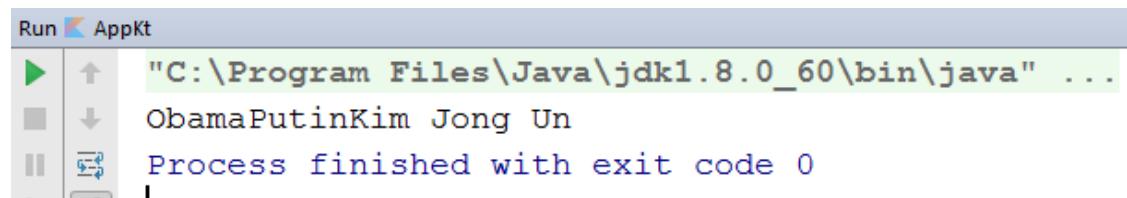
Kết quả:



Hàm print() để xuất dữ liệu trên cùng một dòng, ví dụ:

```
print("Obama")
print("Putin")
print("Kim Jong Un")
```

kết quả:



Ngoài ra Kotlin cũng cung cấp một số ký tự đặc biệt để ta điều hướng cách thức hiển thị dữ liệu, chẳng hạn như:

- \n để Xuống dòng
- \t để thụt đầu dòng
- \" để trích dẫn

Ví dụ ta xuất bài thơ sau:

```
println("Quanh năm buôn bán ở mom sông")
println("Nuôi đủ năm con với một chồng")
println("\tlặng lội thân cò khi quăng vắng")
println("\teo sèo mặt nước buổi đò đồng")
println("Một duyên hai nợ âu đành phận")
println("Năm nắng mười mưa há chẳng công")
println("\tCha mẹ thói đời \"ăn ở bạc\"")
println("\tCó chồng hờ hững cũng nhu không")
```

kết quả:

```
"C:\Program Files\Java\jdk1.8.0_60\bin\java" ...
Quanh năm buôn bán ở mom sông
Nuôi đủ năm con với một chồng
    lặng lội thân cò khi quăng vắng
    eo sèo mặt nước buổi đò đồng
Một duyên hai nợ âu đành phận
Năm nắng mười mưa há chẳng công
    Cha mẹ thói đời "ăn ở bạc"
    Có chồng hờ hững cũng nhu không
```

Như vậy Tui đã hướng dẫn xong cách thức xuất dữ liệu ra màn hình, các bạn thử lại 2 hàm print() và println() này nhé. Chúc các bạn thành công.

Source code tải ở đây: <http://www.mediafire.com/file/wcjeoait1ubpl9a/XuatDuLieu.rar>

Hẹn gặp các bạn ở các bài tiếp theo

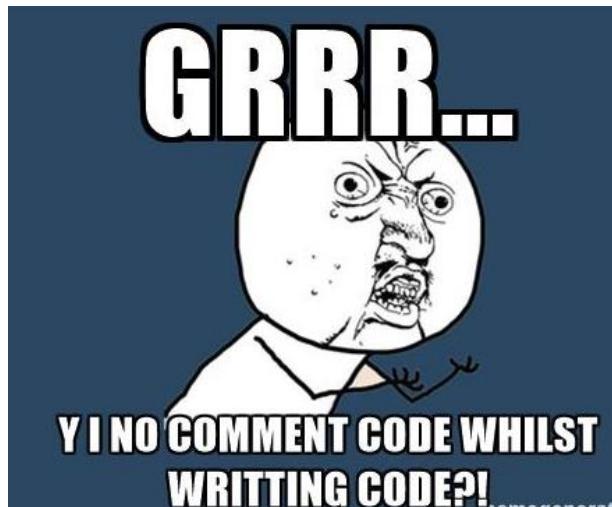
Trần Duy Thành (<http://ssoftinc.com/>)

Bài 5 – Các cách ghi chú quan trọng khi lập trình Kotlin

Tại sao nên ghi chú trong quá trình coding?

Theo quan điểm cá nhân của Tui thì một số lý do sau chúng ta cần phải ghi chú trong quá trình viết mã lệnh:

- Mục đích của ghi chú là giúp giảng giải ngữ nghĩa của các câu lệnh, cũng như chức năng của các hàm, các lớp. Giúp cho các Lập trình viên có thể dễ dàng đọc lại, training nhân viên mới, tạo tài liệu từ các ghi chú
- Các dự án thường làm theo đội, nếu không ghi chú làm sao hiểu? Bản thân ta viết sau 1 thời gian dài cũng sẽ bị quên, phải ghi chú lại để đở tốn tài nguyên nhân lực, thời gian, chi phí...
- Rèn luyện tính chuyên nghiệp trong quá trình làm việc
- Là tinh thần trách nhiệm đối với những vấn đề mình được giao phải hoàn thành, cần được giải thích rõ để khi Say Goodbye người khác vào còng ăn được code của mình



Khi lập trình Kotlin, ta có 3 cách viết ghi chú thường dùng, và những ghi chú này sẽ được bỏ qua trong quá trình biên dịch:

- Ghi chú trên một dòng
- Ghi chú trên nhiều dòng
- Ghi chú theo cú pháp KDoc (KDoc Syntax)

Giờ Ta đi chi tiết vào từng loại ghi chú:

1. Ghi chú trên một dòng ta dùng cú pháp //Ghi chú 1 dòng

Ví dụ:

```
fun main(args: Array<String>) {
    //gọi hàm Cộng 2 số
    val t:Int=Cong(7,8)
    println(t)
}
```

Bạn thấy dòng số 2 có chuỗi //gọi hàm cộng 2 số => đây chính là dòng ghi chú, cú pháp này cho phép ta ghi chú 1 dòng, dựa vào đây ta có thể hiểu lệnh bên dưới dùng để làm gì

2. Ghi chú trên nhiều dòng

Cách ghi chú này sẽ được bao bọc bởi

```
/* ghi chú dòng 1
ghi chú dòng 2
ghi chú dòng n
*/
```

Cho phép ta ghi chú trên nhiều dòng khác nhau.

Ví dụ:

```
/*
Đây là hàm main
dùng để chạy chương trình
*/
fun main(args: Array<String>) {
    //gọi hàm Cộng 2 số
    val t:Int=Cong(7,8)
    println(t)
}
```

Ta quan sát phía trên hàm main có ghi chú nhiều dòng để giải thích chi tiết cho một lệnh, một khối lệnh hay một hàm nào đó (tùy mục đích và văn phong của ta)

3. Ghi chú theo cú pháp KDoc (KDoc Syntax)

Cú pháp này được bao bọc bởi `/** ghi chú dạng Kdoc Syntax */`

Kotlin có tool để generate ra Document, tool này gọi là [Dokka](#), <https://github.com/Kotlin/dokka/blob/master/README.md>

KDoc Syntax có một số Tag ta cần nắm:

`@author` : tác giả

`@sample`: Ví dụ

`@param` : parameter trong hàm

`@return` : kết quả trả về của hàm

Ví dụ:

```
/*
 * @author Trần Duy Thành
 * @param x biến x
 * @param y biến y
 * @return tổng của x và y
 * @sample x=5, y=6 => 11
 * Đây là ghi chú docs
 */
fun Cong(x:Int,y:Int):Int
{
    return x+y
}
/*
Đây là hàm main
dùng để chạy chương trình
*/
fun main(args: Array<String>) {
    //gọi hàm Cộng 2 số
    val t:Int=Cong(7,8)
    println(t)
}
```

Như Vậy Tui đã trình bày xong các cách ghi chú khi lập trình với Kotlin, các bạn chú ý tuân thủ các ghi chú nhé. Hẹn gặp lại các bạn ở những bài tiếp theo, Source code bài này: <http://www.mediafire.com/file/noddk98ug39pn0/HocGhiChu.rar>

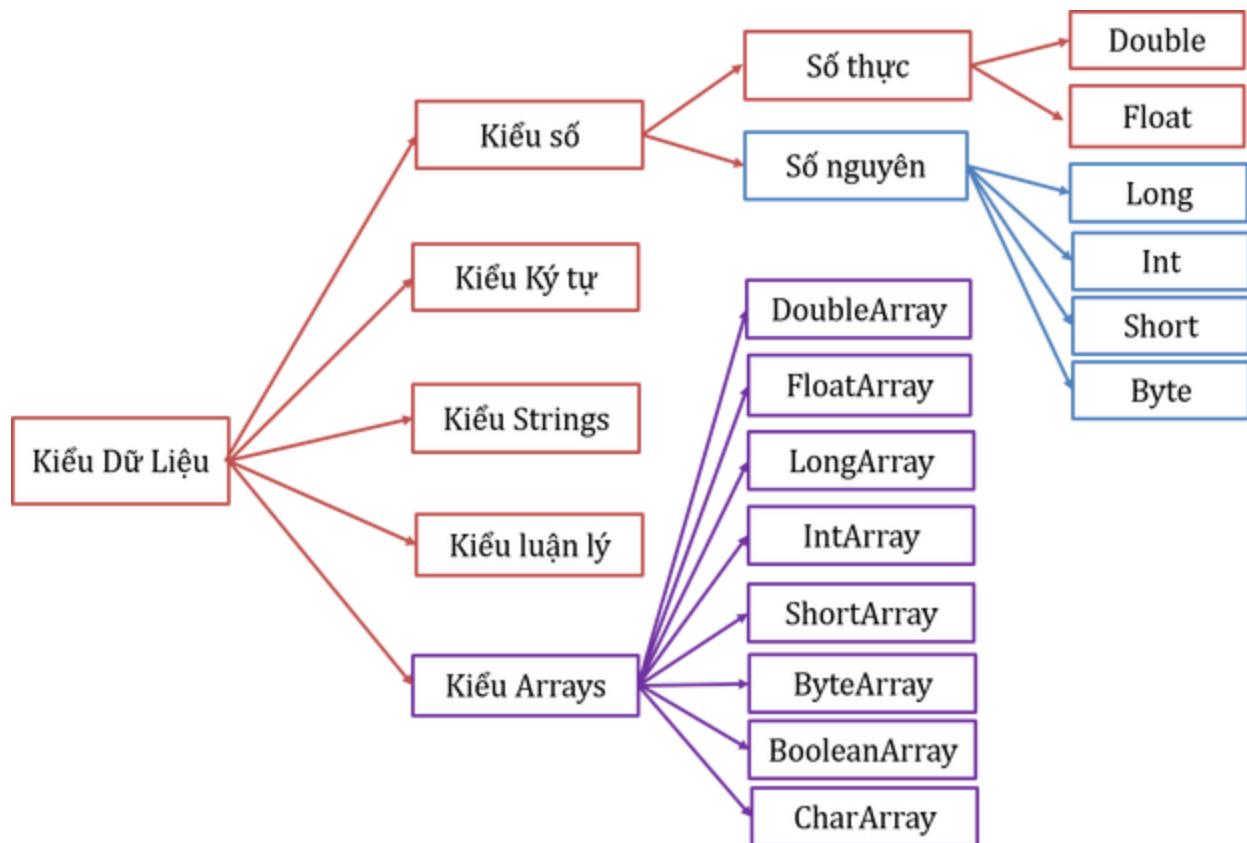
Chúc các bạn thành công

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 6-Kiểu dữ liệu và cách khai báo biến trong Kotlin

Mỗi một ngôn ngữ lập trình đều cung cấp một số kiểu dữ liệu có sẵn để ta lưu trữ xử lý. Kotlin cũng vậy, nó cung cấp hàng loạt kiểu dữ liệu như số thực, số nguyên, ký tự, chuỗi, luận lý và mảng... Có điều là mọi kiểu dữ liệu trong Kotlin đều là hướng đối tượng. Việc nắm chắc ý nghĩa của từng kiểu dữ liệu giúp ta lựa chọn cách khai báo biến có kiểu phù hợp, giúp tối ưu hóa hệ thống.

Dưới đây là hình Tui vẽ tóm tắt quan lại các kiểu dữ liệu được xây dựng sẵn trong Kotlin:



Số thực sẽ có 2 loại đó là Double và Float, Nếu có hằng số để xác định nó là số thực nào thì ta có thể thêm ký tự f hoặc F紧跟 sau hằng số đó:

ví dụ: 113.5 ->số Double , còn 113.5F hoặc 113.5f ->số Float

Số nguyên có 4 loại : Long, Int, Short, Byte . Ta chú ý trường hợp hằng số của Long và Int bằng cách thêm ký tự L

ví dụ: 113->số Int, còn 113L-> số Long (không dùng l thường)

Ta có thể khai báo biến cho các kiểu dữ liệu này như sau:

var tên_bien : Kiểu_Dữ_Liệu=Giá_Trị_Mặc_Định

ví dụ:

```
var x:Long=100L  
var y:Double=113.5  
var f:Float=113.5f  
var i:Int =113  
var s:Short=8  
var b:Byte=1
```

Ta để ý với Kotlin thì không cần thêm dấu chấm phẩy khi kết thúc lệnh

Kiểu ký tự dùng để lưu trữ một ký tự nằm trong nháy đơn:

```
var c:Char='c'
```

Kiểu chuỗi dùng để lưu tập các ký tự được để trong cặp nháy đôi, dùng đối tượng String để khai báo:

```
var ten:String="Trần Duy Thành-0987773061-http://ssoftinc.com/"
```

Ngoài ra ta có thể khai báo chuỗi trên nhiều dòng bằng cách để trong cặp 3 dấu nháy kép:

```
var address:String="""  
số 24 đường 7, khu phố X  
phường 5, Quận 9  
TP.HCM  
"""  
println(address)
```

Kiểu luận lý dùng đối tượng Boolean để khai báo, Kiểu dữ liệu này sẽ lưu trữ 2 giá trị **true** hoặc **false**, kiểu này rất quan trọng, được sử dụng nhiều trong việc kiểm tra các điều kiện:

```
var kq:Boolean=true
```

Với các kiểu dữ liệu Mảng, Kotlin cung cấp cho ta 8 loại mảng ứng với 8 kiểu dữ liệu được built-in trong Kotlin (ngoại trừ String).

Để khai báo và sử dụng kiểu dữ liệu ta làm như sau:

```
var Tên_Mảng: Kiểu_Dữ_Liệu_Mảng=XXXArrayOf (giá trị 1, giá trị 2,..., giá trị n)
```

Với XXX là 8 kiểu dữ liệu tương ứng(viết thường ký tự đầu), ví dụ:

```
var arrX:IntArray= intArrayOf(1,2,3,5)
println(arrX[1])
var arrY:DoubleArray= doubleArrayOf(1.5,2.6,9.0,10.3)
println(arrY[3])
var arrC:CharArray= charArrayOf('a','b','c')
println(arrC[0])
```

Ngoài ra nếu muốn khai báo hàng số thì ta dùng **val** thay vì **var**. **var** cho phép thay đổi giá trị của biến, còn **val** không cho phép. Ta thường nói mutable khi khai báo **var**, readonly khi khai báo **val**.

```
val PI:Double =3.14
x=50
//PI=3.15//không được phép vì PI là readonly
```

Như vậy tới đây Tui đã trình bày xong các kiểu dữ liệu và cách khai báo biến trong Kotlin, các bạn nhớ làm lại bài này nhé để hiểu rõ hơn về kiểu dữ liệu trong Kotlin nhé, cần so sánh sự khác biệt về kiểu dữ liệu cũng như cách khai báo biến so với java.

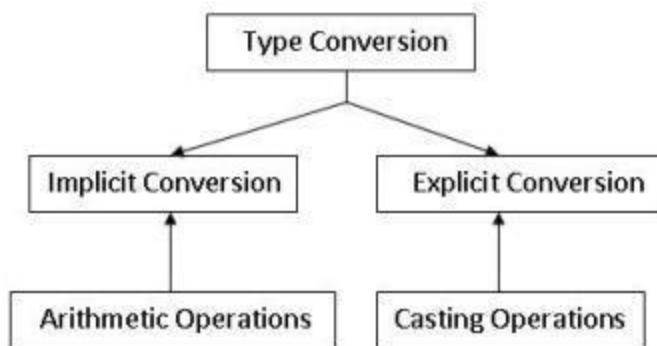
Hẹn gặp lại các bạn ở những bài sau! Source code bài này:
<http://www.mediafire.com/file/2zdwn1pp2e74gc5/HocKieuDuLieu.rar>

Chúc các bạn thành công

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 7 – Ép kiểu dữ liệu trong Kotlin

Trong [bài 6](#) ta đã nắm rõ về các kiểu dữ liệu cũng như cách khai báo biến, trong bài này chúng ta qua phần Ép kiểu dữ liệu trong Kotlin.



Vì sao phải ép kiểu?

Trong quá trình tính toán đôi khi kết quả trả về không còn giống với kiểu dữ liệu chỉ định ban đầu nên ta cần ép kiểu

Khi ép kiểu thường ta gặp 2 trường hợp:

Ép kiểu rộng

Đưa từ kiểu có vùng lưu trữ nhỏ lên kiểu có vùng lưu trữ lớn==>không sợ mất mát dữ liệu.

Ví dụ: Byte=>Short=>Int=>Long=>Float=>Double

Ép kiểu hẹp

Đưa từ kiểu có vùng lưu trữ lớn về kiểu có vùng lưu trữ nhỏ==>có thể bị mất mát dữ liệu

Ví dụ: Double=>Float=>Long=>Int=>Short=>Byte

Vậy thì Ép như thế nào?

Trong Kotlin, bất kỳ kiểu dữ liệu number nào cũng có sẵn các phương thức :

- toByte(): Byte
- toShort(): Short
- toInt(): Int
- toLong(): Long
- toFloat(): Float
- toDouble(): Double
- toChar(): Char

Các phương thức trên thường được gọi là ép kiểu tường minh (chỉ đích danh kiểu dữ liệu nào sẽ được ép về), còn gọi tiếng anh cho nó sang miệng là **Explicit Conversion**.

Ta thử chạy đoạn ép kiểu sau:

```
var X:Int=10
var D:Double=X.toDouble()

println("X=$X")
println("D=$D")
```

Kết quả X=10, và D là 10.0 vì 10 được đưa về số Double là 10.0

Trường hợp này là ép kiểu rộng ==> Đưa từ kiểu có vùng lưu trữ nhỏ hơn lên kiểu dữ liệu có vùng lưu trữ lớn hơn

Xét tiếp ví dụ sau:

```
var F:Double=10.5
var Y:Int = F.toInt()
println("F=$F")
println("Y=$Y")
```

Kết quả F=10.5 và Y=10 ==> bị mất mát dữ liệu, trường hợp này chính là Ép hẹp, đưa từ kiểu dữ liệu có vùng lưu trữ lớn hơn ==> về kiểu dữ liệu có vùng lưu trữ nhỏ hơn . Việc Ép Hẹp này rất nguy hiểm vì nó làm mất mát dữ liệu, đặc biệt các bài toán liên quan tới sai số yêu cầu tối thiểu, hay chuyển khoản ngân hàng. Cần tránh trường hợp này.

Ngoài ra Kotlin còn hỗ trợ một trường hợp là Ép kiểu không tường minh (gọi sang miệng bên tiếng anh là **Implicit Conversion**), tức là Kotlin tự nội suy ra kiểu dữ liệu để gán cho biến, thường do các phép toán số học tạo ra, ví dụ:

```
var t=13L+1
println(t)
```

13L có kiểu LONG, 1 có kiểu Int ==> Kotlin tự lấy kiểu dữ liệu lớn nhất làm chuẩn và gán cho t==>t có kiểu Long

Như vậy Tui đã trình bày xong cách ép kiểu trong Kotlin, bài này rất quan trọng. Chúng ta cần nắm chắc quy tắc ép kiểu để lựa chọn giải pháp ép kiểu cho đúng. Và chắc chắn trong quá trình lập trình ta phải gặp thường xuyên với các trường hợp ép kiểu.

Souce code : <http://www.mediafire.com/file/g6bp1c7og2sqzee/HocEpKieu.rar>

Hẹn gặp lại các bạn ở bài tiếp theo

Chúc các bạn thành công

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 8 – Các toán tử quan trọng trong Kotlin

Mỗi một ngôn ngữ lập trình đều có một tập các toán tử quan trọng để giúp ta tạo các biểu thức trong mã lệnh để giải quyết một số vấn đề liên quan nào đó, và các toán tử này cũng được Kotlin Override thành các phương thức(ta có thể sử dụng các Operator thuận túy và cũng có thể dùng bằng các phương thức):

- Toán tử một ngôi
- Toán tử số học cơ bản
- Toán tử gán
- Toán tử So sánh
- Toán tử Logic
- Toán tử tăng dần giảm dần
- Độ ưu tiên toán tử



Bây giờ ta đi chi tiết vào từng loại toán tử:

1. Toán tử một ngôi:

Toán tử	Mô tả	Phương thức	Ví dụ
+	Số dương	a.unaryPlus()	<code>var a:Int=-8 var b=a.unaryPlus()</code>
-	Số âm	a.unaryMinus()	<code>var a:Int=8 var b=a.unaryMinus()</code>

Ví dụ:

```

fun main(args: Array<String>) {
    var a:Int=-8
    var b=a.unaryPlus()
    var c=a.unaryMinus()
    println("a="+a)
    println("b="+b)
    println("c="+c)
}

```

Chạy đoạn code trên ta sẽ có các kết quả:

```

a=-8
b=8
c=8

```

Chú ý bản thân unaryPlus và unaryMinus không làm thay đổi giá trị nội tại của biến, mà nó trả về một giá trị, ta cần có biến để lưu lại giá trị này.

2. Toán tử số học cơ bản:

Toán tử	Mô tả	Phương thức	Ví dụ
+	Cộng	a.plus(b)	$8 + 5 \Rightarrow 13$ <code>8.plus(5) =>13</code>
-	Trừ	a.minus(b)	$8-5 \Rightarrow 3$ <code>8.minus(5) =>3</code>
*	Nhân	times	$8*5 \Rightarrow 40$ <code>8.times(5) =>40</code>
/	Chia	div	$8/5 \Rightarrow 1$ <code>8.div(5) => 1</code>
%	Chia lấy phần dư	rem	$8 \% 5 \Rightarrow 3$ <code>8.rem(5) =>3</code>

Ví dụ:

```

fun main(args: Array<String>) {
    var a:Int=8
    var b:Int=5

    println("Phép cộng cách 1:" + a + "+" + b + "=" + (a+b))
    println("Phép cộng cách 2:" + a + "+" + b + "=" + (a.plus(b)))
    println("Phép trừ cách 1:" + a + "-" + b + "=" + (a-b))
    println("Phép trừ cách 2:" + a + "-" + b + "=" + (a.minus(b)))
    println("Phép nhân cách 1:" + a + "*" + b + "=" + (a*b))
    println("Phép nhân cách 2:" + a + "*" + b + "=" + (a.times(b)))
    println("Phép chia cách 1:" + a + "/" + b + "=" + (a/b))
    println("Phép chia cách 2:" + a + "/" + b + "=" + (a.div(b)))
    println("Lấy dư cách 1:" + a + "/" + b + "=" + (a%b))
    println("Lấy dư cách 2:" + a + "/" + b + "=" + (a.rem(b)))
    println(8.plus(9))
}

```

Chạy đoạn code trên ta sẽ có các kết quả:

```

Phép cộng cách 1:8+5=13
Phép cộng cách 2:8+5=13
Phép trừ cách 1:8-5=3
Phép trừ cách 2:8-5=3
Phép nhân cách 1:8*5=40
Phép nhân cách 2:8*5=40
Phép chia cách 1:8/5=1
Phép chia cách 2:8/5=1
Lấy dư cách 1:8/5=3
Lấy dư cách 2:8/5=3
17

```

3. Toán tử gán:

Toán tử	Mô tả	Ví dụ	Tương đương với
=	Phép gán giá trị bên phải cho biến bên trái dấu bằng	x=5	
+=	Cộng và gán	x=2 x+=5	x=x+5

		$\Rightarrow x=7$	
$-=$	Trừ và gán	$x=2$ $x-=5$ $\Rightarrow x=-3$	$x=x-5$
$*=$	Nhân và gán	$x=2$ $x*=5$ $\Rightarrow x=10$	$x=x*5$
$/=$	Chia và gán	$x=7$ $x/=5$ $\Rightarrow x=1$	$x=x/5$
$%=$	Chia lấy dư	$x=7$ $x\%=5$ $\Rightarrow x=2$	$x=x\%5$

Ví dụ:

```
fun main(args: Array<String>) {
    var x:Int=5
    x+=2
    println("x=$x")
    x-=2
    println("x=$x")
    x*=2
    println("x=$x")
    x/=2
    println("x=$x")
    x=7
    x\%=3
    println("x=$x")
}
```

Chạy đoạn code trên ta sẽ có các kết quả:

```

x=7
x=5
x=10
x=5
x=1

```

4. Toán tử So sánh:

Toán tử	Mô tả	Phương thức	Ví dụ
==	So sánh bằng	a.equals(b)	$5 == 5 \Rightarrow$ kết quả True
!=	So sánh không bằng	!a.equals(b)	$5 != 5 \Rightarrow$ kết quả False
<	So sánh nhỏ hơn	a.compareTo(b) < 0	$5 < 5 \Rightarrow$ kết quả False
<=	So sánh nhỏ hơn hoặc bằng	a.compareTo(b) <= 0	$5 <= 5 \Rightarrow$ kết quả True
>	So sánh lớn hơn	a.compareTo(b) > 0	$5 > 5.5 \Rightarrow$ kết quả False
>=	So sánh lớn hơn hoặc bằng	a.compareTo(b) >= 0	$113 >= 5 \Rightarrow$ kết quả True

Ví dụ:

```

fun main(args: Array<String>) {
    var a:Int=8
    var b:Int=5
    println(a==b)
    println(a.equals(b))
    println(!a.equals(b))
    println(a.compareTo(b))
    println(3.compareTo(3))
    println(3.compareTo(5))
    println(5.compareTo(3))
}

```

Chạy đoạn code trên ta sẽ có các kết quả:

```
false  
false  
true  
1  
0  
-1  
1
```

5. Toán tử Logic:

Toán tử	Mô tả	Phương thức	Ví dụ
&&	Toán tử Và: Nếu cả hai điều kiện là True thì kết quả sẽ là True	a.and(b)	x=false y=true x&&y==>false
	Toán tử Hoặc: Chỉ cần một điều kiện True thì nó True, tất cả điều kiện False thì nó False	a.or(b)	x=false y=true x y==>true

Ví dụ:

```
fun main(args: Array<String>) {  
    var x:Boolean=true  
    var y:Boolean=false  
    var z:Boolean=false  
    println("x="+x)  
    println("y="+y)  
    println("z="+z)  
    println("x&&y="+(x&&y))  
    println("x.and(y)="+x.and(y))  
    println("x || y ="+(x || y))  
    println("x.or(y)="+x.or(y))  
    println("x || z ="+(x || z))  
    println("x.or(z)="+x.or(z))  
    println("x && z ="+(x && z))
```

```

    println("x.and(z)="+x.and(z))
    println("y || z =" +(y || z))
    println("y.or(z) =" +y.or(z))
    println("y && z =" +(y && z))
    println("y.and(z) =" +y.and(z))
    println("x && y && z =" +(x && y && z))
    println("x.and(y).and(z) =" +x.and(y).and(z))
    println("x||y||z =" +(x||y||z))
    println("x.or(y).or(z) =" +x.or(y).or(z))
}

```

Chạy đoạn code trên ta sẽ có các kết quả:

```

x=true
y=false
z=false
x&&y=false
x.and(y)=false
x // y =true
x.or(y)=true
x // z =true
x.or(z)=true
x && z =false
x.and(z)=false
y // z =false
y.or(z)=false
y && z =false
y.and(z)=false
x && y && z =false
x.and(y).and(z)=false
x//y//z =true
x.or(y).or(z)=true

```

6. Toán tử tăng dần giảm dần

Toán tử	Mô tả	Phương thức	Ví dụ
++	Tăng nội tại biến lên một đơn vị	a.inc()	x=5 x++

			=>x=6
—	giảm nội tại biến xuống một đơn vị	a.dec()	x=5 x--->x=4

Chú ý bản thân hàm inc() và dec() sẽ không làm thay đổi(tăng hay giảm) giá trị nội tại của biến, ta cần có biến khác để lưu giá trị bị thay đổi.

Với toán tử tăng dần và giảm dần này thì việc đặt ++ hay — đặt trước và đặt sau biến có ý nghĩa khác nhau trong một biểu thức phức tạp, thông thường nó sẽ hoạt động theo nguyên tắc (có một số trường hợp đặc biệt phá vỡ nguyên tắc này ta không xét tới vì nó vô cùng hiếm gặp, chủ yếu do Testing Problem):

- Bước 1: Ưu tiên xử lý Prefix trước (các ++ hay — đặt trước biến)
- Bước 2: Tính toán các phép toán còn lại trong biểu thức
- Bước 3: Gán giá trị ở bước 2 cho biến lưu trữ kết quả ở bên trái dấu bằng
- Bước 4: Xử lý Post fix (các ++ hay — đặt sau biến)

Ví dụ:

```
fun main(args: Array<String>) {
    var a:Int=5
    var b:Int=8
    var c:Int=2
    a--
    b++
    var z=a++ + ++b - --c + 7
    println("a="+a)
    println("b="+b)
    println("c="+c)
    println("z="+z)
}
```

Chạy đoạn code trên ta sẽ có các kết quả:

```
a=5
b=10
c=1
z=20
```

Các bạn tự giải thích vì sao ra kết quả ở trên nhé, Tui Busy và cũng chủ ý không giải thích, bạn hãy làm theo 4 bước ở trên thì sẽ ra được kết quả

7. Độ ưu tiên toán tử:

Kotlin có ràng buộc thứ tự ưu tiên của các toán tử. Tuy nhiên tốt nhất là các bạn hay điều khiển nó bằng cách dùng cặp ngoặc tròn () để nó rõ nghĩa hơn. Bảng dưới đây để tham khảo độ ưu tiên từ cao xuống thấp (tuy nhiên có thể quên nó đi mà hãy dùng ngoặc tròn () để chỉ định rõ).

Thứ tự ưu tiên	Toán tử	Miêu tả
1	* / %	Phép nhân, chia, lấy phần dư
2	+ -	Toán tử Cộng, Trừ
3	<= < > >=	Các toán tử so sánh
4	<> == !=	Các toán tử so sánh
5	= %= /= -= += *=	Các toán tử gán
6	&& ,	Các toán tử logic

Như vậy tới đây Tui đã hướng dẫn xong các toán tử thường dùng nhất trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Các bạn có thể tải source code bài này ở đây: <http://www.mediafire.com/file/ty4wjwe0bor55o0/HocToanTu.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 9- Nhập dữ liệu từ bàn phím với Kotlin

Chào các bạn đến với bài học Nhập dữ liệu từ bàn phím với Kotlin, trong bài học này Tui sẽ hướng dẫn các bạn cách thức nhập dữ liệu từ bàn phím như thế nào.

Với Kotlin, để nhập dữ liệu từ bàn phím ta dùng hàm `readLine()`, hàm này nằm trong thư viện mặc định [kotlin.io](#)

Hàm `readLine()` sẽ trả về một chuỗi dữ liệu được nhập vào từ bàn phím hoặc là giá trị `null` nếu như không có dữ liệu.

Từ chuỗi kết quả này ta có thể ép kiểu dữ liệu về bất kỳ kiểu nào mà ta mong muốn ứng với giá trị nhập vào cho phù hợp.



Ví dụ để nhập một chuỗi:

```
fun main(args: Array<String>) {  
    println("Bồ nhí bạn tên gì?:")  
    var ten:String? = readLine()  
    println("Bạn nhập tên:")  
    println(ten)  
}
```

Bạn để ý ở trên Tui khai báo `String?` tức là kiểu dữ liệu có dạng nullable, đây là một cách thức khai báo mới trong Kotlin. Tương tự cho `Int?`, `Double?`

Với một số có định dạng chuỗi trong Kotlin ta luôn luôn chuyển được kiểu dữ liệu về đúng dạng thức ban đầu. `String` trong Kotlin được hỗ trợ tập các phương thức:

Hàm	Mô tả	Ví dụ
toBoolean()	Chuyển chuỗi về Boolean	“true”.toBoolean()
toByte()	Chuyển chuỗi về Byte	“3”.toByte()
toShort()	Chuyển chuỗi về Short	“30”.toShort()
toInt()	Chuyển chuỗi về Int	“15”.toInt()
toLong()	Chuyển chuỗi về Long	“100”.toLong()
toFloat()	Chuyển chuỗi về Float	“15.5”.toFloat()
toDouble()	Chuyển chuỗi về Double	“15.5”.toDouble()

Ví dụ Coding:

```
fun main(args: Array<String>) {
    var x1:Boolean="true".toBoolean()
    println(x1)

    var x2:Byte="3".toByte()
    println(x2)

    var x3:Short="30".toShort()
    println(x3)

    var x4:Int="15".toInt()
    println(x4)

    var x5:Long="100".toLong()
    println(x5)

    var x6:Float="15.5".toFloat()
    println(x6)

    var x7:Double="15.5".toDouble()
    println(x7)
}
```

Chạy đoạn code trên ta sẽ có các kết quả:

```
true  
3  
30  
15  
100  
15.5  
15.5
```

Ví dụ viết chương trình giải phương trình Bậc 1, bài này Tui sẽ áp dụng readLine() nhập liệu từ bàn phím để giải phương trình bậc 1, tuy nhiên Tui sẽ không giải thích cách thức hoạt động của If, Else mà bài sau Tui mới đi vào chi tiết:

```
fun main(args: Array<String>) {  
    var a:Double=0.0  
    var b:Double=0.0  
    println("Nhập a:")  
    var s= readLine()  
    if(s!=null)  
        a=s.toDouble()  
    println("Nhập b:")  
    s= readLine()  
    if(s!=null)  
        b=s.toDouble()  
    if(a==0.0 && b==0.0)  
    {  
        println("Phương trình vô số nghiệm")  
    }  
    else if(a==0.0 && b!=0.0)  
    {  
        println("Phương trình vô nghiệm")  
    }  
    else  
    {  
        var x=-b/a  
        println("No x=" +x)  
    }  
}
```

Chạy đoạn code trên ta sẽ có các kết quả:

```
Nhập a:  
3  
Nhập b:
```

7

No $x = -2.3333333333333335$

Như vậy tới đây Tui đã hướng dẫn xong cách nhập dữ liệu từ bàn phím trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/4456wayewzewxal/HocNhapLieu.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 10- Cấu trúc điều khiển if else trong Kotlin

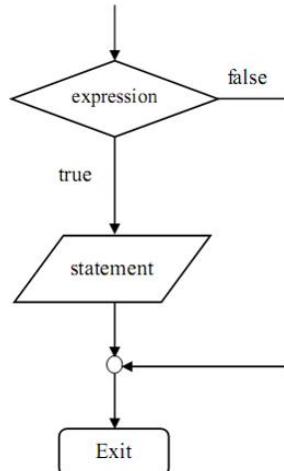
Với Kotlin thì if, else ngoài việc đóng vai trò là một tập lệnh điều kiện (cấu trúc truyền thống), nó còn là một biểu thức trả về giá trị khá thú vị.

1) Ta vào cấu trúc if truyền thống, cú pháp:

```
if  (<expression> )
{
<statement>
}
```

Câu trúc if ở trên chỉ quan tâm tới điều kiện đúng, khi <expression> trong if đúng thì sẽ thực hiện các lệnh <statement> ở bên trong if.

Lưu đồ hoạt động:



Ví dụ, kiểm tra điểm Trung bình ≥ 5 thì ghi đậu:

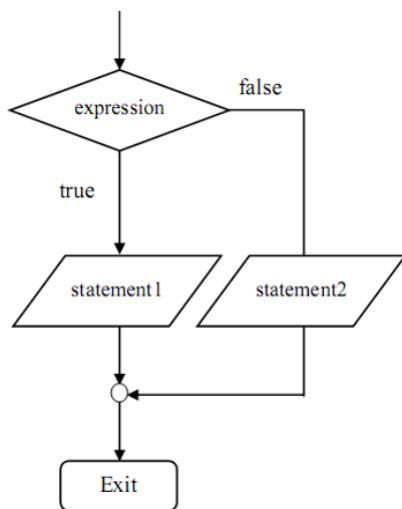
```
fun main(args: Array<String>) {
    var dtb:Double=8.0;
    if(dtb>=5)
    {
        println("Đậu")
    }
}
```

2) Ta vào cấu trúc if và else truyền thống, cú pháp:

```
if (< expression > )
{
< statement 1>
}
else
{
< statement 2>
}
```

Câu trúc if else quan tâm tới điều kiện đúng và sai. Nếu <expression> đúng thì nội dung <statement 1> của if sẽ được thực hiện, nếu <expression> sai thì nội dung <statement 2> của else sẽ được thực hiện. Đây là một trong những câu trúc phổ biến nhất được sử dụng rất nhiều trong quá trình kiểm tra điều kiện.

Lưu đồ hoạt động:



Ví dụ, kiểm tra điểm Trung bình ≥ 5 thì ghi đậu, < 5 thì rớt:

```
fun main(args: Array<String>) {
    var dtb:Double=4.0;
    if(dtb>=5)
    {
        println("Đậu")
    }
    else
    {
        println("Cáu phó")
```

```
    }  
}
```

3) Ta có thể lồng ghép các if else vào với nhau:

Ví dụ: Nhập vào một điểm Trung bình [0..10], kiểm tra điểm này đậu hay rớt:

```
fun main(args: Array<String>) {  
    var dtb:Double=0.0;  
    println("Mời bạn nhập điểm trung bình:")  
    var s:String?= readLine()  
    if(s!=null)  
    {  
        dtb=s.toDouble()  
        if(dtb>=0 && dtb<=10)  
        {  
            if (dtb >= 5)  
            {  
                println("Đậu")  
            }  
            else  
            {  
                println("Rớt")  
            }  
        }  
        else  
        {  
            println("Thím phải nhập điểm [0...10]")  
        }  
    }  
    else  
    {  
        println("Thím nhập lại")  
    }  
}
```

Như vậy ở trên ta đã học được cách tiếp cận if else theo phương pháp truyền thống (như là một biểu thức điều kiện). Nay giờ Tui sẽ trình bày điểm mới if else trong Kotlin đó là nó hoạt động như một biểu thức trả về kết quả:

Ví dụ: cho 2 số a và b, tìm số lớn nhất:

Cách viết theo biểu thức	Cách viết truyền thống
<pre>fun main(args: Array<String>) { var a:Int=10 var b:Int=15 var max:Int max=if (a>b) a else b println("Số lớn nhất =" +max) }</pre>	<pre>fun main(args: Array<String>) { var a:Int=10 var b:Int=15 var max:Int if(a>b) max=a else max=b println("Số lớn nhất = =" +max) }</pre>

Chú ý khi viết if với dạng biểu thức trả về kết quả thì bắt buộc phải có else

Ngoài ra biểu thức if else cũng cho phép viết dạng block lệnh {}, dòng lệnh cuối cùng trong mỗi block đó chính là kết quả trả về, ví dụ:

```
fun main(args: Array<String>) {
    var a:Int=10
    var b:Int=15
    var max = if (a > b) {
        println("Choose a")
        a
    } else {
        println("Choose b")
        b
    }
    println("max =" +max)
}
```

Đoạn lệnh trên khi chạy sẽ ra kết quả:

```
Choose b
max =15
```

Giá trị a ở cuối if, giá trị b ở cuối else là kết quả trả về và gán vào biến max.

Như vậy tới đây Tui đã hướng dẫn xong cấu trúc điều khiển if, else trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt năm được lưu

đồ hoạt động, cách viết theo dạng biểu thức của if else. Các bạn có thể tải source code bài này ở đây: <http://www.mediafire.com/file/6c5swjx84rl9edp/HocIfElse.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 11-Biểu thức when trong Kotlin

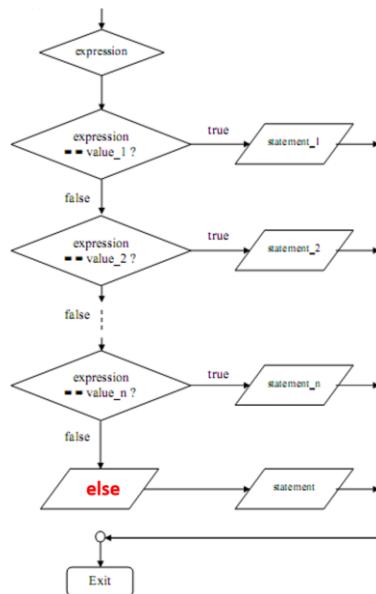
Với Kotlin, biểu thức switch đã được thay thế bởi biểu thức when. When hoạt động rất mạnh mẽ, đa năng, đáp ứng được nhiều trường hợp xử lý cho lập trình viên. Hi vọng với 6 ví dụ chi tiết trong bài này sẽ giúp các bạn nắm chắc về cách thức hoạt động của when, có thể áp dụng nhuần nhuyễn vào các bài thực tế khác phức tạp hơn.

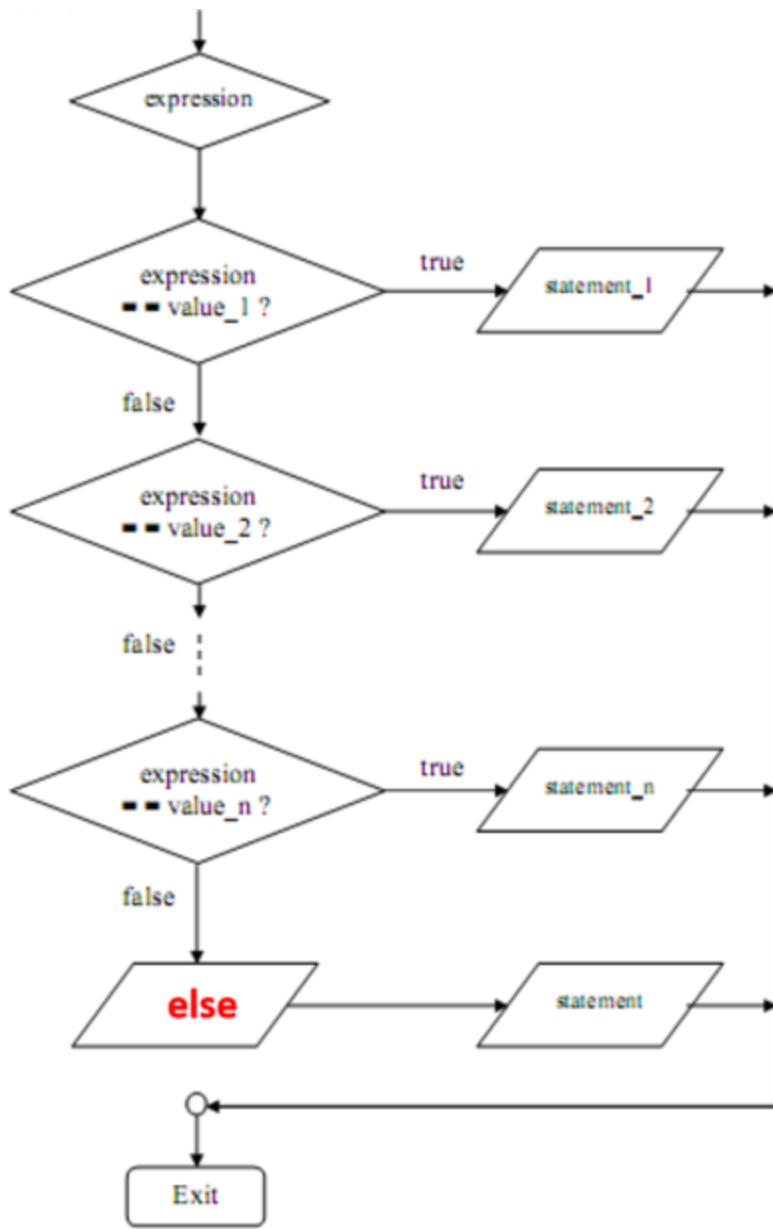
Cú pháp tổng quát của when:

```
when(<expression>){  
    <value 1> -> <statement 1>  
    <value 2> -> <statement 2>  
    else           -> <statement else>  
}
```

when lấy giá trị trong **<expression>**, đếm so sánh với các **<value>** bên trong , nếu trùng khớp với value nào thì **<statement>** đó sẽ được thực thi. Nếu tất cả **<value>** đều không khớp với **<expression>** thì **else** sẽ được thực hiện.

Lưu đồ của **when** có thể được hoạt động như hình dưới đây:





Ví dụ 1:

```

fun main(args: Array<String>) {
    var value:Int=2
    when(value)
    {
        1->println("Hello Obama")
        2->println("Hello Putin")
        3->println("Hello Kim Jong Un")
        else-> println("Hello Everyone!")
    }
}
    
```

Khi chạy đoạn coding trên thì dòng ‘Hello Putin’ sẽ được xuất ra màn hình.

Ví dụ 2: Viết chương trình cho phép nhập vào 2 số a, b và 1 phép toán +,-,*,. Xuất kết quả tương ứng với phép toán

```
fun main(args: Array<String>) {
    var a:Double=0.0;
    var b:Double=0.0;
    var pheptoan:String?
    println("Nhập a:")
    var sa=readLine()
    if(sa!=null)
        a= sa.toDouble()
    println("Nhập b:")
    var sb=readLine()
    if(sb!=null)
        b= sb.toDouble()
    println("Nhập phép toán(+,-,*,/):")
    pheptoan= readLine()
    when(pheptoan)
    {
        "+-> println(a.toString()+" + "+b.toString()+"="+ (a+b) )
        "-> println(a.toString()+" - "+b.toString()+"="+ (a-b) )
        "*-> println(a.toString()+" * "+b.toString()+"="+ (a*b) )
        "/->
            if(b==0.0)
                println("mẫu số phải khác 0")
            else
                println(a.toString() +"/"+b.toString()+"="+ (a/b) )
            else-> println("Bạn nhập lùi")
    }
}
```

Kotlin cũng cho phép ta gom nhóm nhiều điều kiện trong biểu thức `when` nếu như các giá trị này cùng giải quyết một vấn đề nào đó bằng cách dùng dấu phẩy , để ngăn cách các giá trị:

Ví dụ 3: Nhập vào một tháng bất kỳ, hỏi tháng này thuộc quý mấy trong năm. Biết rằng tháng 1,2,3 là quý 1; tháng 4,5,6 là quý 2; tháng 7,8,9 là quý 3; tháng 10,11,12 là quý 4.

```
fun main(args: Array<String>) {
    var month:Int=0
    println("Nhập tháng:")
    var s:String?= readLine()
```

```

if(s!=null)
    month=s.toInt()
when(month)
{
    1,2,3-> println("Tháng "+month+" thuộc quý 1")
    4,5,6-> println("Tháng "+month+" thuộc quý 2")
    7,8,9->println("Tháng "+month+" thuộc quý 3")
    10,11,12->println("Tháng "+month+" thuộc quý 4")
    else-> println("Tháng "+month+" ko hợp lệ")
}
}

```

Cấu trúc **when** của Kotlin rất mạnh mẽ, nó còn cho phép ta kiểm tra theo vùng dữ liệu liên tục:

Ví dụ 4:

```

fun main(args: Array<String>) {
    var tuoi:Int=0
    println("Nhập tuoi:")
    var s:String?= readLine()
    if(s!=null)
        tuoi=s.toInt()
    when(tuoi)
    {
        in 1..5-> println("Tuổi thiếu nhi")
        in 6..9-> println("Tuổi nhi đồng")
        in 10..15-> println("Tuổi thiếu niên")
        in 16..28-> println("Tuổi thanh niên")
        !in 1..100-> println("Không biết tuổi gì")
        else->println("Tuổi già khú dế")
    }
}

```

ở trên bạn thấy Tui để **in 1..5** là cú pháp mà tuoi nằm trong đoạn từ 1 tới 5

!in 1..100 là cú pháp nếu tuoi không nằm trong đoạn 1 tới 100

Như vậy nếu các bạn muốn kiểm tra biến thuộc một vùng dữ liệu liên tục nào đó thì có thể dùng cú pháp ở trên, Tui thấy nó rất hay trong trường hợp này. Ta có thể áp dụng tốt vào các giải thuật so khớp.

Ngoài ra **when còn là một biểu thức trả về kết quả**, cách viết linh động như sau:

Ví dụ 5:

```
fun main(args: Array<String>) {
    var x:Int=8
    var kq=when (x)
    {
        in 1..10->x+100
        in 20..30->x-1000
        else -> x
    }
    println(kq)
}
```

Chạy lên ta sẽ có $kq=108$. Vì cách viết trên sẽ gán kết quả vào cho biến kq , khi $x=8$ thì thửa vùng $1..10$ suy ra kết quả $kq=8+100=108$

Cuối cùng **when** có thể hoạt động như chuỗi biểu thức if-else if:

Ví dụ 6:

```
fun main(args: Array<String>) {
    var x:Int=100
    when
    {
        x%2==0-> println("X là số chẵn")
        x%2!=0->println("X là số lẻ")
    }
}
```

Lệnh trên chính là if else, chạy lên sẽ xuất X là số chẵn. Ta có thể viết when như trên để thay thế cho các if else.

Với biến x ở ví dụ trên(hoặc biến bất kỳ), nếu bạn muốn xuất giá trị của biến cùng với chuỗi thì nên dùng ký tự Đô La đằng trước biến, ví dụ: \$x thì Kotlin sẽ tự tìm tới tên biến để xuất giá trị của nó ra, ta có thể viết lại lệnh trên như sau:

```
fun main(args: Array<String>) {
    var x:Int=100
    when
    {
        x%2==0-> println("$x là số chẵn")
        x%2!=0->println("$x là số lẻ")
```

```
}
```

Khi chạy ta sẽ có kết quả: **100 là số chẵn**

Như vậy tới đây Tui đã hướng dẫn xong cấu trúc **when** trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt nǎm được lưu đồ hoạt động, cách viết theo dạng when khác nhau.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/6mxazxl8726t1df/HocWhen.rar>

Hẹn gặp các bạn ở những bài tiếp theo

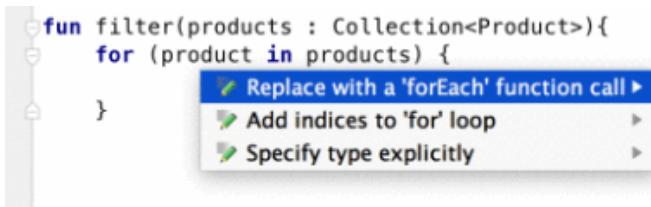
Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 12-Vòng lặp for trong Kotlin

Với mọi ngôn ngữ lập trình thì các cấu trúc vòng lặp rất là quan trọng, đặc biệt là vòng for rất là phổ biến và hầu như trong mọi phần mềm đều chắc chắn sử dụng để giải quyết những công việc lặp đi lặp lại một cách có quy luật.

Kotlin cung cấp khoảng 5+ cách thức làm việc với vòng lặp for rất đa dạng, các kỹ thuật này Tui sẽ trình bày chi tiết thông qua việc giải thích lý thuyết cũng như các ví dụ minh họa, hi vọng các bạn sẽ hiểu thật chắc Vòng lặp for trong Kotlin.



1) Loại for thứ 1 – Duyệt tuần tự hết giá trị trong danh sách (**closed range**)

```
for (i in a..b)
{
    xử lý biến i
}
```

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động tăng dần từ a cho tới b. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 1: Viết chương trình tính giá thừa của một số nguyên dương n:

```
fun main(args: Array<String>) {
    var gt:Int=1
    val n:Int=5
    for (i in 1..n)
    {
        gt *= i
    }
    println("$n != $gt")
}
```

Ở đoạn lệnh trên, i sẽ chạy tuần tự từ 1 tới n (với n=5). Mỗi lần chạy nó sẽ nhân dồn i cho biến gt. Kết thúc vòng lặp ta sẽ được giá thừa của n.

Chi tiết quá trình chạy tính giai thừa theo thuật giải ở trên:

Lần lặp	Giá Trị Biến i	Giá Trị Biến gt
0		gt=1
1	1	gt*=i=>gt=1*1=1
2	2	gt*=i=>gt=1*2=2
3	3	gt*=i=>gt=2*3=6
4	4	gt*=i=>gt=6*4=24
5	5	gt*=i=>gt=24*5=120

Sau đó lệnh:

```
println("$n!=$gt")
```

Được thực hiện, với kết quả: **5!=120**

2) Loại for thứ 2 – Duyệt tuần tự gần hết giá trị trong danh sách (**half-open range**)

```
for (i in a until b)
{
    Xử lý biến i
}
```

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động tăng dần từ a cho tới gần b. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 2: Viết chương trình tính tổng từ 1 tới gần số nguyên dương n:

```
fun main(args: Array<String>) {
    var sum:Int=0
    val n:Int=5
    for (i in 1 until n)
    {
        sum += i
    }
}
```

```
    println("Tổng=$sum")
```

```
}
```

Chi tiết quá trình tính tổng:

Lần lặp	Giá Trị Biến i	Giá Trị Biến sum
0		sum=0
1	1	sum+=i=>sum=0+1=1
2	2	sum+=i=>sum=1+2=3
3	3	sum+=i=>sum=3+3=6
4	4	sum+=i=>sum=6+4=10

Chú ý là chỉ chạy tới 4, vì theo cú pháp của `until` là chạy tới gần bằng. Do đó ở trên `until n` tức là gần bằng `n`, ở đây `n=5 => i` chạy tới 4.

Sau đó lệnh:

```
println("Tổng=$sum")
```

Được thực hiện, với kết quả: `Tổng=10`

3) Loại for thứ 3 – Điều hướng bước nhảy step

```
for (i in a .. b step x)
{
    Xử lý biến i
}
```

Với cú pháp ở trên thì biến `i` thực ra là biến bước nhảy, nó tự động tăng dần từ `a` cho tới `b`, nhưng mỗi lần duyệt nó tăng theo `x` đơn vị. Ta có thể thay tên biến `i` thành tên biến bất kỳ

Ví dụ 3: Viết chương trình tính tổng các số chẵn nhỏ hơn hoặc bằng số nguyên dương `n`:

```

fun main(args: Array<String>) {
    var sum:Int=0
    var n:Int=10
    for (i in 2 .. n step 2)
        sum+=i
    println("Tổng chẵn=$sum")
}

```

Chi tiết quá trình tính tổng chẵn:

lần lặp	Giá Trị Biến i	Giá Trị Biến sum
0		sum=0
1	2	sum+=i =>sum=0+2=2
2	4	sum+=i =>sum=2+4=6
3	6	sum+=i =>sum=6+6=12
4	8	sum+=i =>sum=12+8=20
5	10	sum+=i =>sum=20+10=30

Chương trình chạy mỗi lần tăng i lên 2 đơn vị, ta khởi tạo i chạy từ 2 nên nó sẽ tự động tăng thành các số chẵn, gấp 10 thì nó thực hiện và kết thúc vòng lặp.

Sau đó lệnh:

```
println("Tổng chẵn=$sum")
```

Được thực hiện, với kết quả: **Tổng chẵn =30**

4) Loại for thứ 4 – Điều hướng bước nhảy downTo

```

for (i in b downTo a)
{
    Xử lý biến i
}

```

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động giảm dần từ b cho tới a, nhưng mỗi lần duyệt nó giảm 1 đơn vị. Ta có thể thay tên biến i thành tên biến bất kỳ

hoặc

```
for (i in b downTo a step x)
{
    Xử lý biến i
}
```

Với cú pháp ở trên thì biến i thực ra là biến bước nhảy, nó tự động giảm dần từ b cho tới a, nhưng mỗi lần duyệt nó giảm x đơn vị. Ta có thể thay tên biến i thành tên biến bất kỳ

Ví dụ 4: Viết chương trình tính Uớc số chung lớn nhất của 2 số bất kỳ

```
fun main(args: Array<String>) {
    var a:Int=9
    var b:Int=6
    var ucscln=1
    var min;if (a>b) b else a
    for (i in min downTo 1)
    {
        if(a%i==0 && b%i==0)
        {
            ucscln=i
            break
        }
    }
    println("USCL của $a và $b = $ucscln")
}
```

Chi tiết quá trình tìm ước số chung lớn nhất:

trước khi vào vòng lặp for thì ta được các thông tin sau:

a=9, b=6, ucscln=1, min=6 (ta phải tìm min vì với 2 số thì may ra nó chỉ chia hết cho số nhỏ hơn)

Lần lặp	Giá Trị Biến i	Giá Trị Biến ucscln
0		a=9, b=6, ucscln=1, min=6
1	6	if (a%i==0 && b%i==0) <=>if (9%6==0 && 6%6==0)

		=>không thỏa
2	5	<pre>if (a%i==0 && b%i==0) <=>if (9%5==0 && 6%5==0) =>không thỏa</pre>
3	4	<pre>if (a%i==0 && b%i==0) <=>if (9%4==0 && 6%4==0) =>không thỏa</pre>
4	3	<pre>if (a%i==0 && b%i==0) <=>if (9%3==0 && 6%3==0) =>thỏa điều kiện =>thực hiện ucscln=i=>ucscln=3 =>break =>thoát khỏi vòng lặp for</pre>

Tới lần lặp thứ 4 thì lúc này cả a và b đều chia hết cho i ($i=3$) nên nội dung if sẽ được thực hiện, lúc này $ucscln=3$ và lệnh break ngay sau đó thực hiện ngừng vòng lặp

Sau đó lệnh:

```
println("USCL của $a và $b = $ucscln")
```

Được thực hiện, với kết quả: **USCL của 9 và 6 = 3**

5) Loại for thứ 5 – Lặp tập đối tượng

```
for (item in collection)
{
    println(item)
}
```

Cấu trúc for trên sẽ duyệt từng đối tượng trong một tập đối tượng

Ví dụ 5: Duyệt danh sách tên sản phẩm

```
fun main(args: Array<String>) {
    var dsSanPham= arrayOf("kotlin","java", "c#", "python", "R")
    for (item in dsSanPham)
        println(item)
}
```

Đoạn code trên, chương trình sẽ duyệt tuần tự từng phần tử trong mảng dsSanPham và xuất tên của chúng ra.

Ngoài ra ta cũng có thể duyệt theo vị trí, từ vị trí này ta có thể xuất ra giá trị như sau:

```
fun main(args: Array<String>) {
    var dsSanPham= arrayOf("kotlin","java", "c#", "python", "R")
    for (i in dsSanPham.indices)
        println("Sản phẩm thứ $i có tên "+dsSanPham[i])
}
```

Cuối cùng kotlin cũng hỗ trợ vừa lấy vị trí vừa lấy giá trị theo cách sau:

```
fun main(args: Array<String>) {
    var dsSanPham= arrayOf("kotlin","java", "c#", "python", "R")
    for ((index,value) in dsSanPham.withIndex())
    {
        println("Sản phẩm thứ $index có tên $value")
    }
}
```

Như vậy tới đây Tui đã hướng dẫn xong vòng lặp **for** trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt nǎm được 5 cách thức vận hành của for để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/7ku4okq72mj1iam/HocFor.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 13-Vòng lặp while trong Kotlin

Ở [bài 12](#) chúng ta đã nắm chắc được cách thức hoạt động của vòng lặp for, trong bài này Tui sẽ hướng dẫn các bạn cấu trúc lặp với while. Đây cũng là một trong những cấu trúc lặp khá phổ biến trong các ngôn ngữ lập trình không riêng gì Kotlin, cú pháp:

```
while (expression)
{
    statement
}
```

Các bước thực hiện:

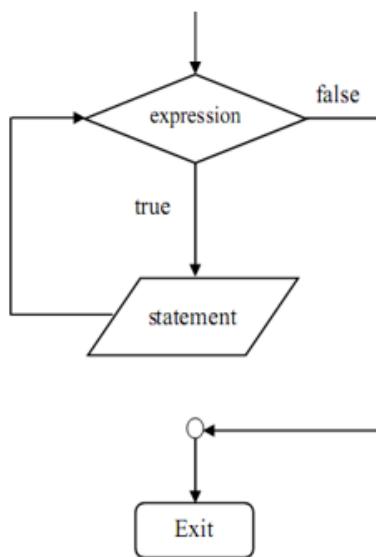
- B1: Expression được định trị
- B2: Nếu kết quả là **true** thì statement thực thi và quay lại B1
- B3: Nếu kết quả là **false** thì thoát khỏi vòng lặp while.

Để thoát vòng lặp: dùng **break**

Để di chuyển sớm qua lần lặp tiếp theo : dùng **continue**

Lưu ý: Lệnh trong while có thể không được thực hiện lần nào do ngay từ đầu expression không thỏa

Lưu đồ hoạt động:



Ví dụ 1: Viết chương trình tính 5 giai thừa

```
fun main(args: Array<String>) {
    var n:Int = 5
    var gt:Int=1
    var i:Int = 1
    while (i <= n)
    {
        gt *= i
        i++
    }
    println("$n! = $gt")
}
```

Giải thích chi tiết quá trình chạy:

Khởi tạo n=5, gt=1, i=1

Lần 1: kiểm tra $i \leq n \Leftrightarrow 1 \leq 5 \Rightarrow$ đúng \Rightarrow làm body while:

$gt = gt * i = 1 * 1 = 1$

$i++ \Rightarrow i = i + 1 = 1 + 1 = 2$

Lần 2: kiểm tra $i \leq n \Leftrightarrow 2 \leq 5 \Rightarrow$ đúng \Rightarrow làm body while:

$gt = gt * i = 1 * 2 = 2$

$i++ \Rightarrow i = i + 1 = 2 + 1 = 3$

Lần 3: kiểm tra $i \leq n \Leftrightarrow 3 \leq 5 \Rightarrow$ đúng \Rightarrow làm body while:

$gt = gt * i = 2 * 3 = 6$

$i++ \Rightarrow i = i + 1 = 3 + 1 = 4$

Lần 4: Kiểm tra $i \leq n \Leftrightarrow 4 \leq 5 \Rightarrow$ đúng \Rightarrow làm body while:

$gt = gt * i = 6 * 4 = 24$

$i++ \Rightarrow i = i + 1 = 4 + 1 = 5$

Lần 5: Kiểm tra $i \leq n \Leftrightarrow 5 \leq 5 \Rightarrow$ đúng \Rightarrow làm body while:

$gt = gt * i = 24 * 5 = 120$

$i++ \Rightarrow i = i + 1 = 5 + 1 = 6$

Lần 6: kiểm tra $i \leq n \Leftrightarrow 6 \leq 5 \Rightarrow$ sai \Rightarrow ngừng while

Kết thúc chương trình ta được giao thura =120

Ta có thể kết hợp các vòng lặp khác nhau lại, ví dụ có thể kết hợp vòng while và for, ví dụ dưới đây sẽ minh họa việc kết hợp này:

Ví dụ 2: Viết chương trình Kiểm tra một số bất kỳ có phải là số nguyên tố hay không(số nguyên tố là số chỉ chia hết cho 1 và chính nó, số 0 và số 1 không phải là số nguyên tố), chương trình cho phép hỏi người dùng có muốn tiếp tục nữa hay không?

```
fun main(args: Array<String>) {
    var n:Int=0
    var s:String?
    println("Chào mừng đến với Chương trình kiểm tra số Nguyên
Tố")
    while (true)
    {
        println("Nhập số nguyên: ")
        s= readLine()
        if(s!=null)
            n=s.toInt()
        var dem:Int=0
        for(i in 1..n)
        {
            if(n%i==0)
                dem++
        }
        if(dem==2)
            println("$n là số nguyên tố")
        else
            println("$n ko phải là số nguyên tố")
        print("Tiếp không?(c/k) : ")
        s= readLine()
        if(s=="k")
            break;
    }
    println("Chào tạm biệt")
}
```

Chạy chương trình và nhập các giá trị sau để xem kết quả:

```
Chào mừng đến với Chương trình kiểm tra số Nguyên Tố
Nhập số nguyên:
5
```

5 là số nguyên tố

Tiếp không?(c/k):c

Nhập số nguyên:

7

7 là số nguyên tố

Tiếp không?(c/k):c

Nhập số nguyên:

6

6 ko phải là số nguyên tố

Tiếp không?(c/k):k

Chào tạm biệt

Như vậy tới đây Tui đã hướng dẫn xong vòng lặp while trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt nắm được cách thức vận hành của while để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/gv9pj5v721374qr/HocWhile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 14-Vòng lặp do while trong Kotlin

Ở [bài 13](#) chúng ta đã hiểu được cấu trúc lặp while, trong bài này chúng ta qua một cấu trúc tương tự đó là do...while

Cú pháp vòng lặp do...while:

```
do
{
    statement
}
while(expression)
```

Các bước thực hiện:

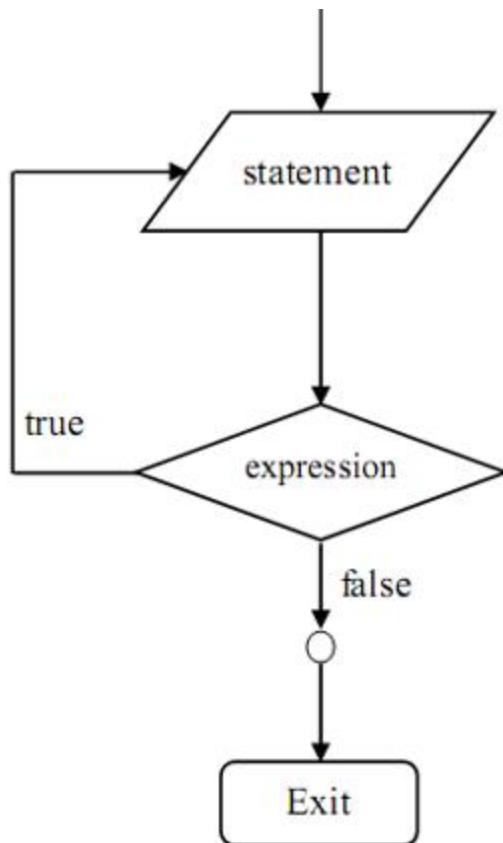
- B1:Statement được thực hiện
- B2:Expression được định trị.
- Nếu expression là true thì quay lại bước 1
- Nếu expression là false thì thoát khỏi vòng lặp.

Để thoát vòng lặp: dùng **break**

Để di chuyển sớm qua lần lặp tiếp theo : dùng **continue**

Lưu ý: **Lệnh trong do...while chắc chắn được thực hiện ít nhất một lần.**

Lưu đồ hoạt động:



Ví dụ 1: Viết chương trình tính 5 giai thừa

```

fun main(args: Array<String>) {
    var n:Int = 5
    var gt:Int=1
    var i:Int = 1
    do
    {
        gt *= i
        i++
    }while (i<=n)
    println("$n! =$gt")
}
  
```

Giải thích chi tiết quá trình chạy:

Khởi tạo: n=5; gt=1, i=1

Lần 1: gt=gt*i=1*1=1

$i++ \Rightarrow i = i + 1 = 1 + 1 = 2$

Kiểm tra $i \leq n \Leftrightarrow 2 \leq 5 \Rightarrow$ đúng

Lần 2: $gt = gt * i = 1 * 2 = 2$

$i++ \Rightarrow i = i + 1 = 2 + 1 = 3$

Kiểm tra $i \leq n \Leftrightarrow 3 \leq 5 \Rightarrow$ đúng

Lần 3: $gt = gt * i = 2 * 3 = 6$

$i++ \Rightarrow i = i + 1 = 3 + 1 = 4$

Kiểm tra $i \leq n \Leftrightarrow 4 \leq 5 \Rightarrow$ đúng

Lần 4: $gt = gt * i = 6 * 4 = 24$

$i++ \Rightarrow i = i + 1 = 4 + 1 = 5$

Kiểm tra $i \leq n \Leftrightarrow 5 \leq 5 \Rightarrow$ đúng

Lần 5: $gt = gt * i = 24 * 5 = 120$

$i++ \Rightarrow i = i + 1 = 5 + 1 = 6$

Kiểm tra $i \leq n \Leftrightarrow 6 \leq 5 \Rightarrow$ sai \Rightarrow ngừng do while

Kết thúc chương trình ta được giao thừa = 120

Tương tự như các vòng lặp khác, do..while cũng có thể kết hợp lồng ghép các vòng lặp với nhau:

Ví dụ 2: Viết chương trình kiểm tra 1 năm bất kỳ có phải năm nhuận hay không (Năm nhuận là năm chia hết cho 4 nhưng không chia hết cho 100 hoặc chia hết cho 400). Chương trình bắt buộc phải nhập năm ≥ 0 , nếu nhập sai bắt nhập lại cho tới khi nào nhập đúng, kết thúc chương trình cho phép người dùng lựa chọn tiếp tục hay không:

```
fun main(args: Array<String>) {
    var year:Int=0
    var s:String?
    println("Chương trình kiểm tra năm nhuận:")
    do
    {
```

```

println("Nhập 1 năm:")
s= readLine()
while (s==null || s.toInt()<0)
{
    println("Nhập sai năm, nhập lại:")
    s= readLine()
}
year=s.toInt()
if (year%4==0 && year%100!=0 || year%400==0)
{
    println("Năm $year là năm nhuận")
}
else
{
    println("Năm $year ko phải là năm nhuận")
}
print("Tiếp không?(c/k):")
s= readLine()
if (s=="k")
    break;
}while (true)
println("Tạm biệt")
}

```

Chạy chương trình và nhập các giá trị sau để xem kết quả:

```

Chương trình kiểm tra năm nhuận:
Nhập 1 năm:
-4
Nhập sai năm, nhập lại:
2000
Năm 2000 là năm nhuận
Tiếp không?(c/k):c
Nhập 1 năm:
2015
Năm 2015 ko phải là năm nhuận
Tiếp không?(c/k):kTạm biệt

```

Như vậy tới đây Tui đã hướng dẫn xong vòng lặp **do...while** trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt năm được cách thức vận hành của do..while để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/zehz2biw992vxeh/HocDoWhile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

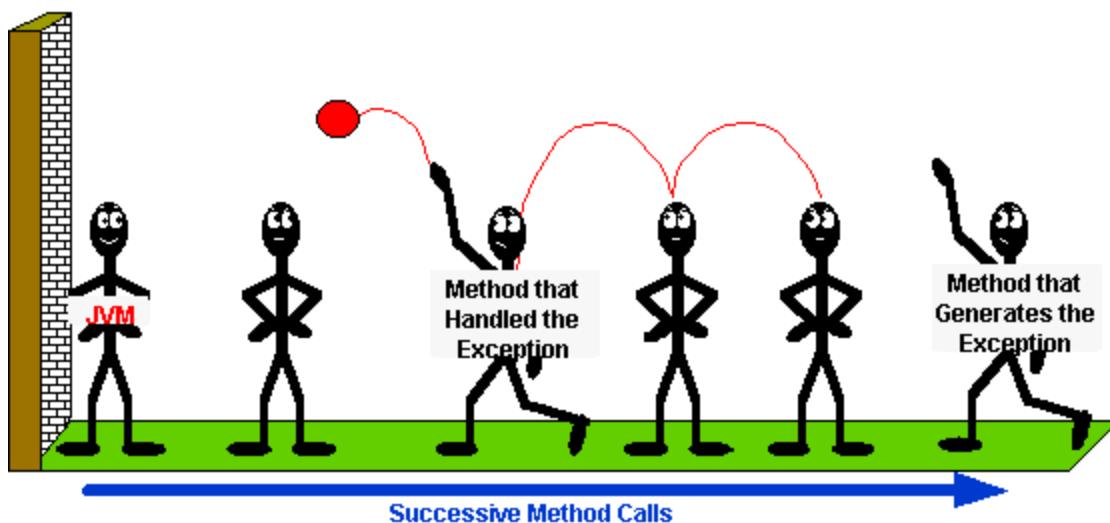
Trần Duy Thành (<http://ssoftinc.com/>)

Bài 15-Xử lý biệt lệ trong Kotlin

Tại sao phải xử lý biệt lệ?

Trong quá trình thực thi phần mềm sẽ có những lỗi phát sinh mà trong quá trình coding ta đã dự đoán hoặc chưa dự đoán được

Việc kiểm soát các biệt lệ giúp cho phần mềm tiếp tục hoạt động nếu lỗi xảy ra hoặc cũng đưa ra các gợi ý bên phía User Problem



Khi lập trình thường chúng ta gặp 3 cấp độ lỗi:

- Lỗi biên dịch (mới học lập trình, viết đâu sai đó... rất đáng thương)
- Lỗi runtime exception (học lập trình chưa thông, chạy lên báo lỗi om sòm)
- Lỗi logic exception – sai nghiệp vụ yêu cầu (đã thành lập trình viên, gặp những lỗi khó sửa)

Và khi gặp lỗi thì thường có 2 hành vi với lỗi: Không quan tâm (Unchecked error) và quan tâm (Checked error)

Để hướng tới lập trình viên chuyên nghiệp thì ta cần quan tâm tới những lỗi này, phải kiểm tra cẩn thận để khi có gặp lỗi xảy ra thì chương trình vẫn tiếp tục mà không bị tắt ngang.

Kotlin hỗ trợ chúng ta cú pháp tổng quát để xử lý biệt lệ như sau:

```

try {
    // viết lệnh ở đây và các lệnh này có khả năng sinh ra lỗi
}

catch (e: SomeException) {
    // handler lỗi ở đây -> thông báo lỗi chi tiết để biết mà sửa cái gì
}

finally {
    // optional finally block – cho dù có lỗi hay không có lỗi xảy ra thì block luôn luôn thực hiện
}

```

Ví dụ 1: Viết chương trình chia 2 số a và b. Chủ ý cho mẫu số =0 để sinh ra lỗi

```

fun main(args: Array<String>) {
    try {
        var a:Int=5
        var b:Int=0;
        var c=a/b
        println("$a/$b=$c")
    }
    catch (e:Exception)
    {
        println(e.message)
    }
    finally {
        println("Đây là finally, 100% chạy, cho dù lỗi hay
không")
    }
}

```

Khi chạy đoạn lệnh ở trên thì đến dòng a/b sẽ sinh ra lỗi và nó nhảy vào exception, sau đó finally sẽ được thực hiện:

```

/by zero
Đây là finally, 100% chạy, cho dù lỗi hay không

```

Ngoài ra Kotlin cũng hỗ trợ từ khóa **throws** để ném lỗi này ra nơi khác, những chỗ nào gọi nó sẽ xử lý lỗi này.

Ví dụ 2:

```
fun chia(a:Int,b:Int):Int
{
    if(b==0)
        throw Exception("Mẫu số =0")
    return a/b
}
fun main(args: Array<String>) {
    chia(5,0)
    println("Cám ơn!")
}
```

Khi chạy chương trình ở ví dụ 2 lên thì nó sẽ xuất ra thông báo lỗi: Mẫu số =0 và tắt ngang phần mềm, không cho dòng chữ “Cám ơn” có cơ hội xuất hiện. Vì lúc này ta chưa dùng try..catch để checked error:

```
Exception in thread "main" java.lang.Exception: Mẫu số =0
at App_bietle_throwsKt.chia(app_bietle_throws.kt:7)
at App_bietle_throwsKt.main(app_bietle_throws.kt:11)
```

Ta sửa lại mã lệnh ở trên để khi có lỗi xảy ra thì chương trình vẫn tiếp tục hoạt động:

Ví dụ 3:

```
fun chia(a:Int,b:Int):Int
{
    if(b==0)
        throw Exception("Mẫu số =0")
    return a/b
}
fun main(args: Array<String>) {
    try {
        chia(5, 0)
    }
    catch (e:Exception)
    {
        println(e.message)
    }
}
```

```
    println("Cám ơn!")
}
```

Khi có try .. catch thì cho dù có lỗi xảy ra, chương trình vẫn tiếp tục thực hiện các lệnh còn lại (tránh tắt ngang phần mềm), chương trình trên khi chạy sẽ có kết quả:

```
Mẫu số =0  
Cám ơn!
```

Như vậy tới đây Tui đã hướng dẫn xong xử lý biệt lệ trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Đặc biệt năm được cách thức vận hành của try..catch..finally...throw để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/ko46c0s2d476669/HocXuLyBietLe.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 16-Cách gỡ lỗi Kotlin bằng công cụ Debug

Ở [bài 15](#) Tui đã trình bày cách xử lý biệt lệ để kiểm soát lỗi phát sinh trong quá trình runtime. Tuy nhiên để lần vết tường tận từng bước chương trình thực hiện để tìm ra các lỗi khó như lỗi Logic thì bắt buộc ta phải sử dụng công cụ Debug, phần mềm IntelliJ IDEA đã tích hợp sẵn công cụ Debug giúp chúng ta dễ dàng lần vết.

Theo kinh nghiệm của Tui thì thời gian viết mã lệnh mới chiếm khoảng 40% còn 60% thời gian còn lại là chúng ta phải lần vết sửa lỗi (nói cho sang miệng thì giới lập trình viên gọi bằng cụm từ [Fix Bug](#)). Dĩ nhiên con số này chỉ là cảm tính của Tui thôi, nó không phải là 1 con số chính xác cho từng lập trình viên. Nhưng Tui cam đoan rằng thời gian Fix Bug phải luôn luôn > thời gian viết code mới.

Với một công cụ Debug bất kỳ thì ta thường có 3 thao tác chính:

- Cách đặt break point
- Cách debug từng dòng
- Cách xem giá trị của biến trong khi debug

Bây giờ Tui sẽ hướng dẫn chi tiết từng thao tác trong công cụ IntelliJ IDEA để bạn Debug nhé, giả sử Tui có đoạn lệnh giải phương trình bậc 2 dưới đây (Tui đang chủ ý làm sai Logic), nhiệm vụ của ta là tìm ra các lỗi sai Logic này bằng công cụ Debug:

```
fun NhapGiaTri():Double
{
    var s:String?= readLine()
    if(s!=null)
        return s.toDouble()
    return 0.0
}
fun main(args: Array<String>) {
    var a:Double=0.0
    var b:Double=0.0
    var c:Double=0.0
    println("Nhập a:")
    a=NhapGiaTri()
    println("Nhập b:")
    b=NhapGiaTri()
    println("Nhập c:")
    c=NhapGiaTri()
```

```

if(a==0.0)
{
    if(b==0.0 && c==0.0)
        println("PT VÔ SỐ NÓ")
    else if(b==0.0 && c!=0.0)
        println("PT VÔ NÓ")
    else
        println("X=" + (-b/c))
}
else
{
    val delta:Double=Math.pow(b, 2.0)-4*a*c
    if(delta<0)
        println("VÔ NGHIỆM")
    else if(delta==0.0)
    {
        var x:Double=-b/(2*a)
        println("NÓ KÉP X1=X2=$x")
    }
    else
    {
        var x1:Double=(b-Math.sqrt(delta))/(2*a)
        var x2:Double=(b+Math.sqrt(delta))/(2*a)
        println("X1=$x1")
        println("X2=$x2")
    }
}
}

```

Nhìn lệnh trên bạn phát hiện ra sai Logic chỗ nào?

Tui đưa ra 2 trường hợp để Test xuất hiện lỗi Logic bên trên như sau:

1.Trường hợp 1: $0x^2+5x+2=0$

Nếu đúng thì kết quả phải là $x=-0.4$, nhưng khi chạy lên kết quả lại là $x=-2.5$ đây chính là lỗi sai Logic đầu tiên

2.Trường hợp 2: $x^2+5x-6 =0$

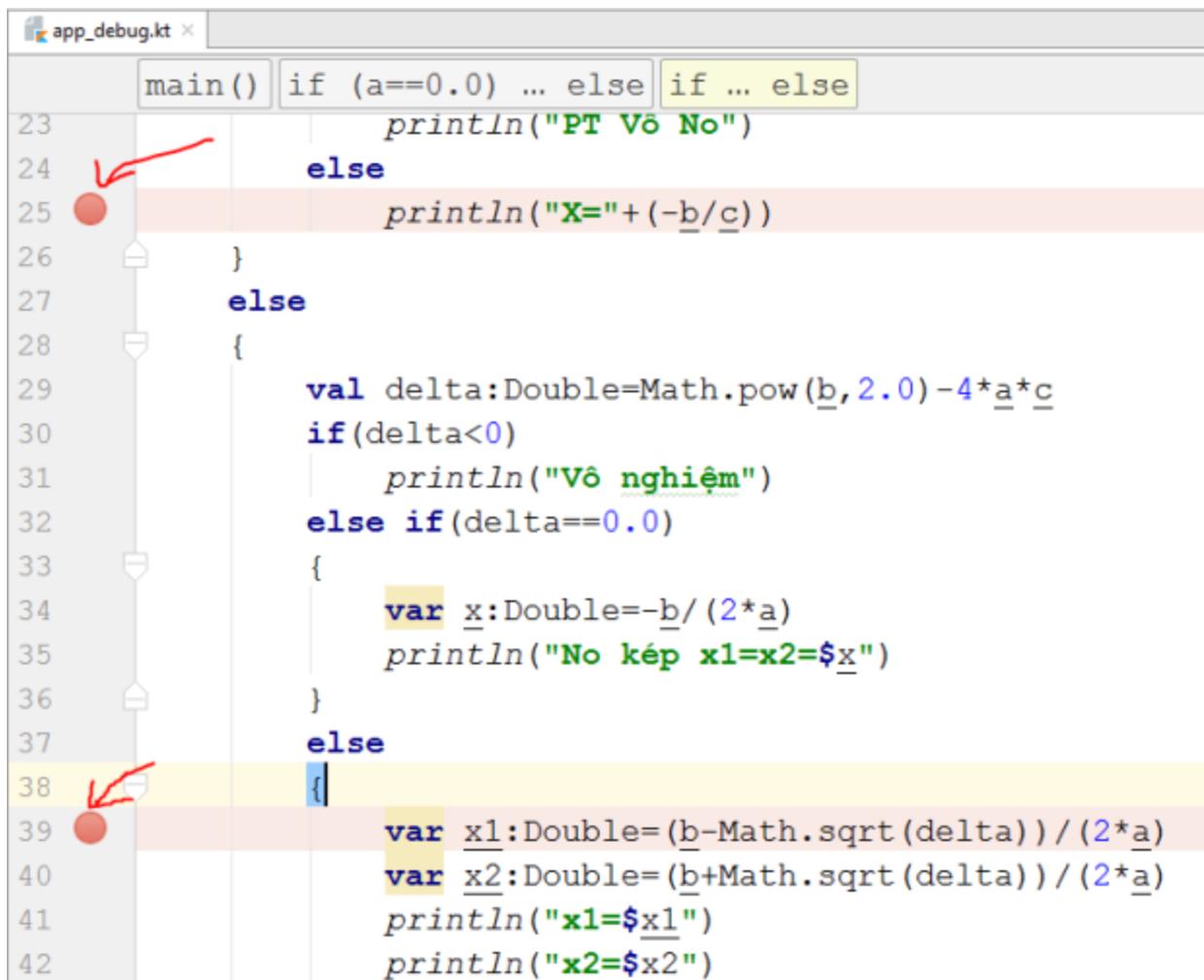
Nếu đúng thì kết quả sẽ là: $x_1=-6$, $x_2=1$. Nhưng kết quả ở trên lại ra $x_1=-1.0$, $x_2=6.0$ đây chính là lỗi sai Logic thứ 2

Vậy làm sao sửa cho đúng? thường ta phải Debug vào từng dòng để kiểm tra (đĩ nhiên khi làm thực tế thì chả bao giờ gặp những giải thuật tầm thường như thế này, đây là Tui lấy ví dụ cụ thể bài giải phương trình bậc 2 để hướng dẫn các bạn cách thức Debug và cũng chỉ rõ luôn 2 chỗ sai Logic) .

Bây giờ ta đi vào chi tiết:

Bước 1: Muốn dừng lại kiểm tra lệnh ở dòng nào thì tạo break point ở dòng đó (bước này gọi là **đặt Break Point**):

Ví dụ bạn muốn đặt Break Point ở dòng lệnh 25 và dòng lệnh 39 thì lần lượt bấm chuột vào kẽ bên số dòng chỗ mũi tên màu đỏ Tui trỏ tới đó, nó sẽ xuất hiện Núm tròn màu đỏ, muốn bỏ nó thì Bấm lại lần nữa:

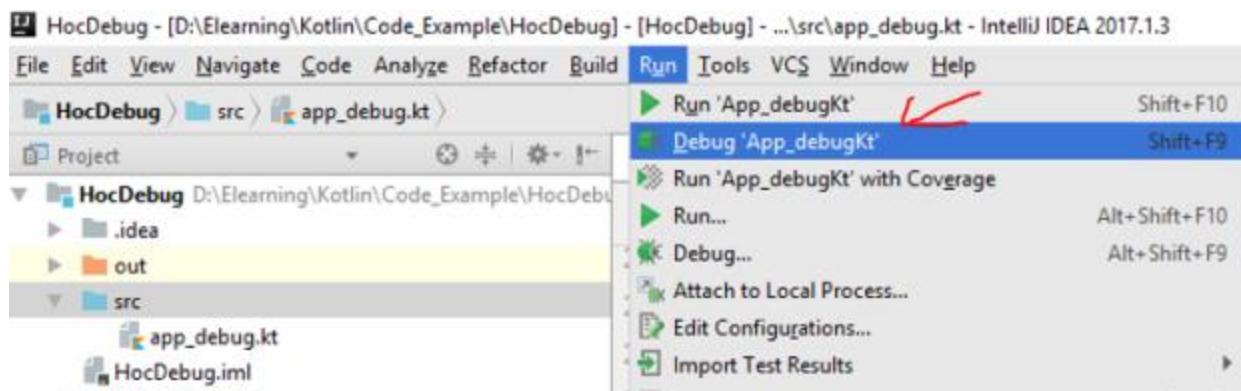


```
app_debug.kt
main() if (a==0.0) ... else if ... else
23         println("PT Vô Óc")
24     else
25         println("X=" + (-b/c))
26     }
27 else
28 {
29     val delta:Double=Math.pow(b,2.0)-4*a*c
30     if(delta<0)
31         println("Vô nghiệm")
32     else if(delta==0.0)
33     {
34         var x:Double=-b/(2*a)
35         println("Óc kép x1=x2=$x")
36     }
37     else
38     {
39         var x1:Double=(b-Math.sqrt(delta))/(2*a)
40         var x2:Double=(b+Math.sqrt(delta))/(2*a)
41         println("x1=$x1")
42         println("x2=$x2")
```

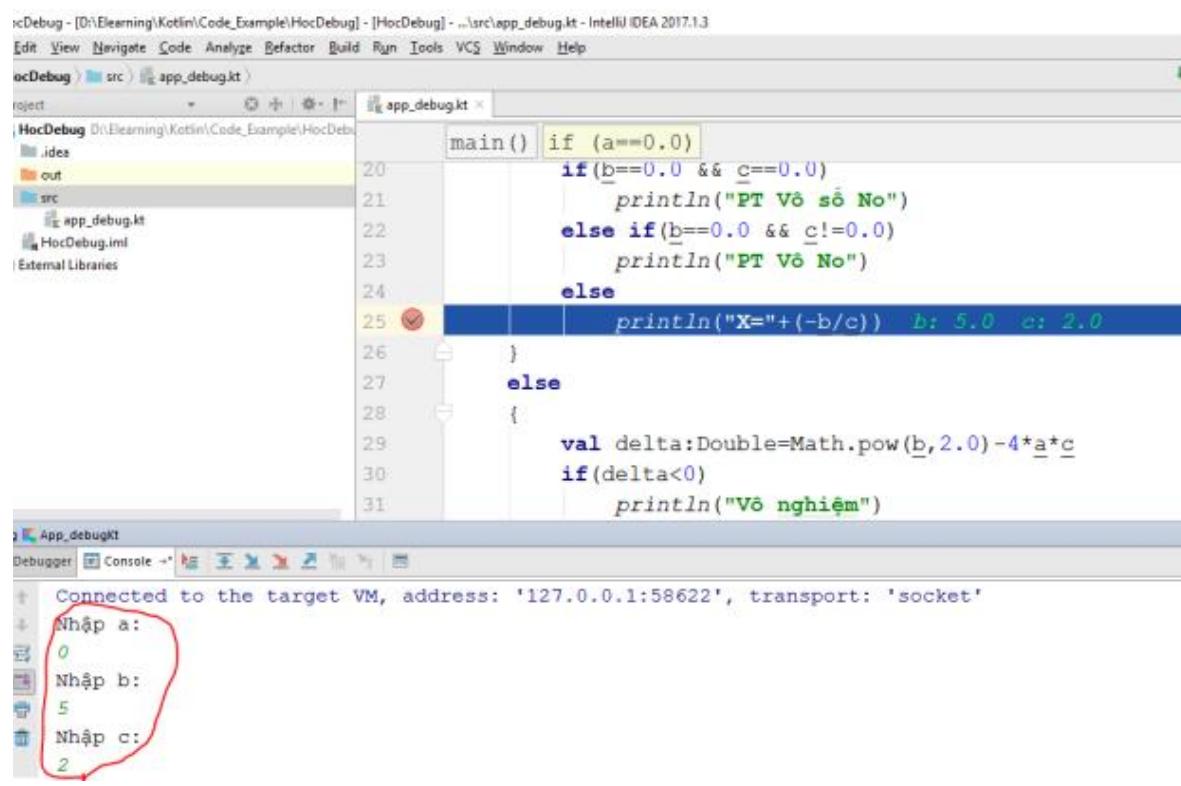
Khi dòng nào có Núm tròn đỏ xuất hiện thì chính là Break Point được đặt thành công.

Bước 2: Chạy Debug từng dòng

Ta vào menu Run/ chọn Debug App_debug.kt (là tập tin mà ta viết lệnh main muốn Debug)



Chương trình sẽ thực thi và ta nhập các giá trị cho các biến:



Ở trên ta thử nhập $a=0$, $b=5$ và $c=2 \Rightarrow$ lúc này chương trình sẽ tự động xuất hiện Tab Debugger kề bên Tab Console và di chuyển con trỏ lệnh tới dòng 25 mà ta đã đặt Break Point:

```

HocDebug - [D:\Elearning\Kotlin\Code_Example\HocDebug] - [HocDebug] - ...src\app_debug.kt - IntelliJ IDEA 2017.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HocDebug src app_debug.kt
Project .idea out src app_debug.kt HocDebug.iml External Libraries
main() if (a==0.0)
20     if(b==0.0 && c==0.0)
21         println("PT VÔ SỐ NÓ")
22     else if(b==0.0 && c!=0.0)
23         println("PT VÔ NÓ")
24     else
25         println("X=" + (-b/c)) b: 5.0 c: 2.0
26     }
27     else
28     {
29         val delta:Double=Math.pow(b, 2.0)-4*a*c
30         if(delta<0)
31             println("VÔ NGHIỆM")

```

Debug App_debugKt

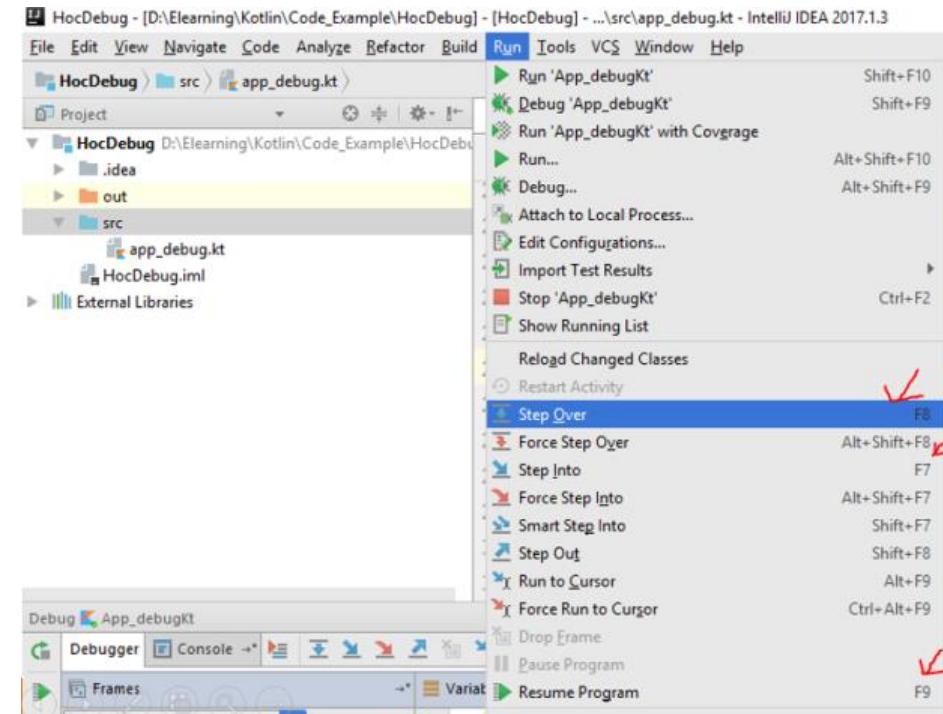
Frames Variables

main@ in group "m..." main25, App.debugKt

a = 0.0
b = 5.0
c = 2.0

Ở màn hình trên ta có thể thấy được giá trị của các biến ngay dòng debug.

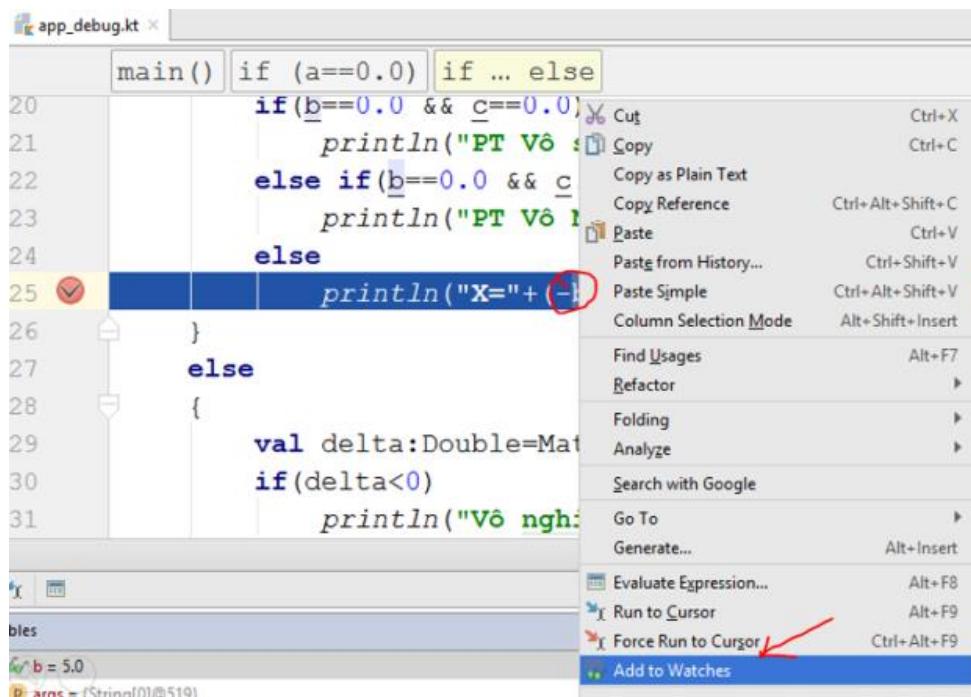
Và công cụ ở chế độ Debug cho phép ta dùng các phím F7, F8, F9 để điều hướng quá trình Debug:



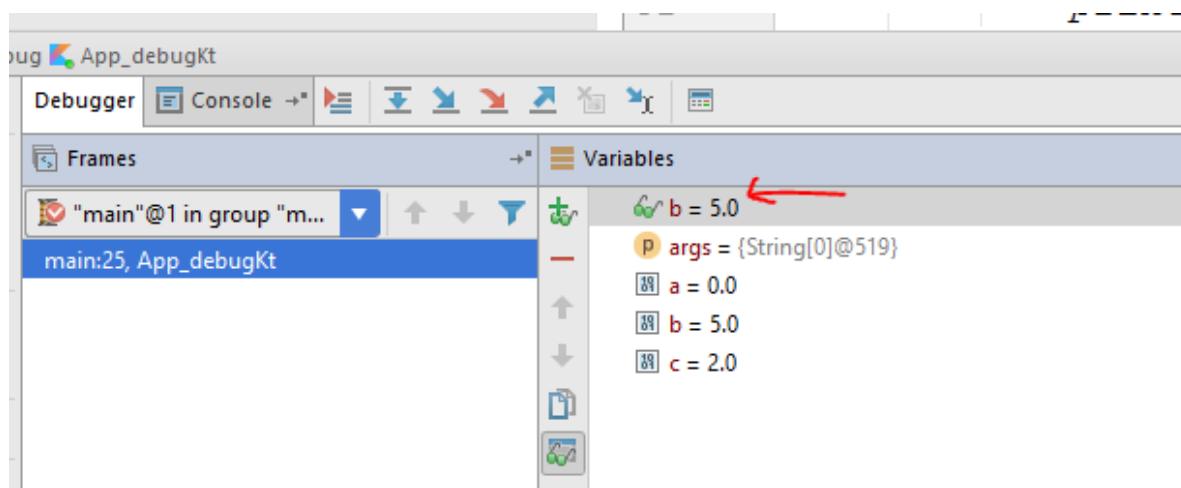
F7 là đi vào bên trong phương thức được triệu gọi, F8 là vượt qua phương thức triệu gọi, F9 là quay trở về chế độ bình thường. Và còn nhiều các phím tắt khác các bạn có thể test.

Bước 3: Cách xem giá trị của biến trong khi debug

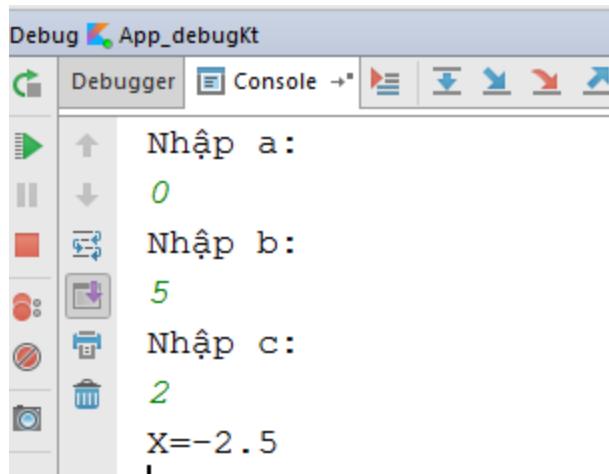
Để xem thêm các giá trị của các Biến trong quá trình Debug, IntelliJ IDEA còn cung cấp chức năng Add to Watches (bôi đen biến bất kỳ → bấm chuột phải → chọn Add to Watches):



Sau khi chọn Add to Watches → biến này sẽ được xuất hiện trong màn hình Variables:



Ta nhấn F8 để chạy tiếp thì thấy kết quả:



Dĩ nhiên vì lỗi Logic Tui nói trước nên tới đây ta cũng biết kết quả nó sai, và cần sửa lại lệnh cho đúng (thực tế nó khó hơn nhiều, có khi bạn phải F7 F8 cả ngày chưa chắc tìm ra nó lỗi chỗ nào mà sửa)

Tương tự như vậy bạn Test trường hợp thứ 2 là có 2 nghiệm phân biệt bị sai

Như vậy tới đây Tui đã hướng dẫn xong cách gỡ lỗi Kotlin bằng công cụ Debug, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Càn thành thạo công cụ Debug, các phím duyệt debug cũng như add to Watches để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code sai logic bài này ở đây:

<http://www.mediafire.com/file/zo2t52p0qorz1dz/HocDebug.rar>

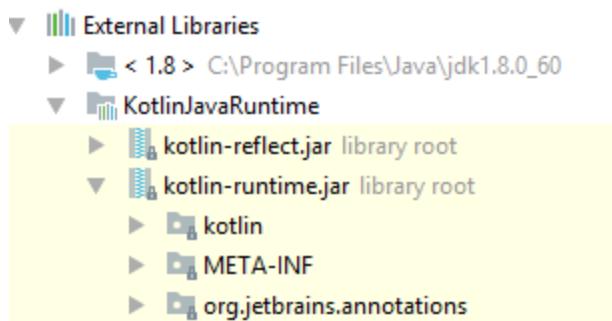
Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 17-Các thư viện quan trọng thường dùng trong Kotlin

Ngôn ngữ nào cũng có tập các thư viện giúp ta giải quyết nhanh những công việc nào đó, và chắc chắn chúng ta phải sử dụng vì ta không thể tự viết ra được, ta phải biết cách sử dụng nó. Các thư viện này có thể nằm trong JVM hay KotlinJavaRuntime hay ở bất kỳ thư viện ngoài nào khác:



Trong bài này Tui sẽ trình bày một số thư viện thường dùng như:

- Thư viện xử lý dữ liệu số
- Thư viện xử lý ngày tháng
- Thư viện xử lý toán học
- Thư viện xử lý số ngẫu nhiên
- Thư viện xử lý chuỗi

Và còn rất nhiều thư viện xử lý khác nữa, khi nào gặp ta lại ngâm cứu tiếp

Các thư viện này có thể được sử dụng từ JVM, Bây giờ Tui đi vào chi tiết từng thư viện xử lý

- **Thư viện xử lý dữ liệu số**

Muốn định dạng chữ số thập phân cho số thực ta có thể dùng format sau:

```
fun main(args: Array<String>) {  
    var d:Double=10.0/3.0  
    println(d)  
    println("%.2f".format(d))  
}
```

Kết quả xuất ra màn hình khi chưa định dạng và đã định dạng thập phân:

```
3.333333333333335
```

```
3.33
```

“%.2f” là cú pháp định số thập phân, ở trong Tui để số 2 tức là 2 số thập phân, bạn muốn bao nhiêu số thì tự thay thế.

Ta có thể sử dụng thư viện DecimalFormat nằm trong JVM để định dạng số, ví dụ dưới đây minh họa tạo ngăn cách phần nghìn:

```
import java.text.DecimalFormat
import java.text.DecimalFormatSymbols
import java.util.Locale

/**
 * Created by cafe on 28/05/2017.
 */
fun main(args: Array<String>) {
    var x:Int=986553823
    var dcf=DecimalFormat("#,###")
    var dcfs=DecimalFormatSymbols(Locale.getDefault())
    dcfs.groupingSeparator=','
    dcf.decimalFormatSymbols=dcfs
    println(x)
    println(dcf.format(x))
}
```

Kết quả khi chạy chương trình:

```
986553823
```

```
986,553,823
```

Chú ý những dòng lệnh import ở trên là IntelliJ IDEA tự import giúp, ta chỉ cần gõ Control + Space khi dùng các thư viện thì tự động nó xuất hiện. Kotlin giúp ta triệu gọi thư viện trong JVM một cách dễ dàng nhất giúp các lập trình viên Java dễ chuyển đổi ngôn ngữ

- **Thư viện xử lý ngày tháng**

Khi xử lý ngày tháng năm ta thường dùng 3 thư viện:

Date
Calendar
SimpleDateFormat

Các thư viện này nằm trong gói:

```
import java.util.Date  
import java.util.Calendar  
import java.text.SimpleDateFormat
```

- Lấy ngày tháng năm hiện tại:

```
var cal=Calendar.getInstance()
```

- Lấy từng tiêu chí

```
var year=cal.get(Calendar.YEAR)
```

```
var month=cal.get(Calendar.MONTH)
```

```
var day=cal.get(Calendar.DAY_OF_MONTH)
```

- Thay đổi thông số ngày tháng năm:

```
cal.set(Calendar.YEAR, 1990);
```

```
cal.set(Calendar.MONTH,2)
```

```
cal.set(Calendar.DAY_OF_MONTH,20)
```

- Lấy đối tượng ngày tháng năm:

```
var date=cal.time
```

- Để định dạng ngày tháng năm dd/MM/YYYY

```
var date=cal.time  
var sdf=SimpleDateFormat("dd/MM/yyyy");  
println(sdf.format(date))
```

Ví dụ:

```

import java.util.Date
import java.util.Calendar
import java.text.SimpleDateFormat

/**
 * Created by cafe on 28/05/2017.
 */
fun main(args: Array<String>) {
    var cal=Calendar.getInstance()
    var year=cal.get(Calendar.YEAR)
    var month=cal.get(Calendar.MONTH)
    var day=cal.get(Calendar.DAY_OF_MONTH)
    println("Năm=$year")
    println("Tháng=$month")
    println("Ngày=$day")

    var date=cal.time
    var sdf=SimpleDateFormat("dd/MM/yyyy");
    println(sdf.format(date))
    var sdf2=SimpleDateFormat("dd/MM/yyyy hh:mm:ss aaa");
    println(sdf2.format(date))
}

```

Kết quả khi chạy chương trình:

```

Năm=2017
Tháng=4
Ngày=28
28/05/2017
28/05/2017 07:26:33 AM

```

Chú ý tháng luôn chạy từ 0->11 nên khi ra số 4 chính là tháng 5 (ta có thể xem SimpleDateFormat xuất ra đúng tháng) .

`hh:mm:ss aaa` là định dạng giờ:phút:giây, `aaa` là AM hoặc PM . Muốn định dạng 24h thì thay `hh` thành `HH`

- **Thư viện xử lý toán học**

Để xử lý toán học ta dùng thư viện Math nằm trong gói `java.lang`

Một số phương thức thường dùng của Math:

Tên phương thức	Mô tả
PI	Trả về giá trị PI
abs(a)	Trả về trị tuyệt đối của a
max(a,b)	Trả về giá trị lớn nhất giữa a và b
min(a,b)	Trả về giá trị nhỏ nhất giữa a và b
sqrt(a)	Trả về căn bậc 2 của a
pow(x,y)	Tính lũy thừa x^y
sin(radian)	Tính sin, radian=Math.PI*góc/180
cos(radian)	Tính cos
tan(radian)	Tính tan

Ví dụ:

```
fun main(args: Array<String>) {
    println("Số PI=" + Math.PI)
    println("Giá trị tuyệt đối của -4=" + Math.abs(-4))
    println("số " + Math.max(9, 2) + " là số lớn")
    println("Căn bậc 2 của 25=" + Math.sqrt(25.0))
    println("3 mũ 4 =" + Math.pow(3.0, 4.0))
    var goc=45
    var rad=Math.PI*goc/180
    println("sin($goc)=" + Math.sin(rad))
    println("cos($goc)=" + Math.cos(rad))
    println("tan($goc)=" + Math.tan(rad))
    println("cotan($goc)=" + Math.cos(rad) / Math.sin(rad))
}
```

Kết quả khi chạy chương trình:

```
Số PI=3.141592653589793
Giá trị tuyệt đối của -4=4
số 9 là số lớn
Căn bậc 2 của 25=5.0
3 mũ 4 =81.0
sin(45)=0.7071067811865475
```

```
cos(45)=0.7071067811865476  
tan(45)=0.9999999999999999  
cotan(45)=1.0000000000000002
```

- **Thư viện xử lý số ngẫu nhiên**

Để xử lý số ngẫu nhiên ta dùng lớp Random nằm trong gói import `java.util.Random`

```
var rd=Random()
```

```
var x=rd.nextInt(n);
```

Trả về số ngẫu nhiên từ [0...n-1]

Ví dụ:

```
[0....100]=>rd.nextInt(101)
```

```
[-100 ...100]=>-100+rd.nextInt(201)
```

```
[-100 ... -50]=>-100+rd.nextInt(51)
```

`rd.nextDouble()` trả về số ngẫu nhiên [0...1)

Ví dụ viết Game đoán số theo mô tả:

Máy tính sẽ ra 1 số ngẫu nhiên [0..100], yêu cầu người chơi đoán số này, cho phép đoán sai 7 lần (quá 7 lần thì Game Over). Nếu đoán sai thì phải cho người chơi biết là số Người chơi đoán nhỏ hơn hay lớn hơn số của máy.

=> Sau khi kết thúc game (WIN or LOST)=> hỏi xem Người chơi có muốn chơi nữa không.

```
import java.util.Random  
  
/*  
 * Created by cafe on 28/05/2017.  
 */  
fun main(args: Array<String>) {  
    while (true) {  
        choi()  
    }  
}
```

```

        println("Chơi nữa không thím? (c/k) :")
        val tl = readLine()
        if (tl.equals("k", ignoreCase = true)) {
            break
        }
        println("Tạm biệt Thím!")
    }
}

fun choi()
{
    val rd = Random()
    val soMay = rd.nextInt(101)
    println("Máy đã ra 1 số [0...100] mời Thím đoán!")
    var soNguoi: Int=0
    var soLanDoan = 0
    do {
        println("Bạn đoán số gì?:")
        var s=readLine()
        if(s!=null)
            soNguoi = s.toInt()
        soLanDoan++
        println("Thím đoán lần thứ " + soLanDoan)
        if (soNguoi == soMay) {
            println("Chúc mừng Thím! Thím đoán đúng, số máy =" +
soMay)
            break
        }
        if (soNguoi < soMay) {
            println("Thím đoán sai! số máy > số thím")
        } else {
            println("Thím đoán sai! số máy <số thím")
        }
        if (soLanDoan == 7) {
            println("Thím đã Cáo Phó, vì đoán 7 lần mà ko
trúng!")
            break
        }
    } while (soLanDoan <= 7)
}

```

Kết quả khi chạy chương trình:

Máy đã ra 1 số [0...100] mời Thím đoán!
Bạn đoán số gì?:
50
Thím đoán lần thứ 1

```
Thím đoán sai! số máy > số thím
Bạn đoán số gì?: 
75
Thím đoán lần thứ 2
Thím đoán sai! số máy <số thím
Bạn đoán số gì?: 
62
Thím đoán lần thứ 3
Chúc mừng Thím! Thím đoán đúng, số máy =62
Chơi nữa không thím? (c/k) :
k
Tạm biệt Thím!
```

- **Thư viện xử lý Chuỗi**

Ta thường sử dụng lớp **StringBuilder** (nằm trong `kotlin.text`) để xử lý chuỗi

- ✓ **StringBuilder()**: Mặc định tạo ra một đối tượng StringBuilder có thể lưu giữ được 16 ký tự
- ✓ **StringBuilder(int capacity)**: Tạo ra một đối tượng StringBuilder có thể lưu giữ được capacity ký tự
- ✓ **StringBuilder(String s)**: Tạo một đối tượng StringBuilder lấy thông tin từ chuỗi s

Các phương thức thường sử dụng:

- ✓ `append()` – nối chuỗi
- ✓ `insert()` – chèn chuỗi
- ✓ `delete()` – xóa chuỗi
- ✓ `reverse()` – đảo ngược chuỗi

Ví dụ:

```
fun main(args: Array<String>) {
    var sb= StringBuilder("Obama Putin")
    println(sb.toString())
    sb.insert(5,"Kim Jong Un")
    println(sb.toString())
    sb.append(" Donald Trump")
    println(sb.toString())
    sb.reverse()
    println(sb.toString())
}
```

Kết quả khi chạy chương trình:

Obama Putin
ObamaKim Jong Un Putin
ObamaKim Jong Un Putin Donald Trump
pmurT dlanoD nituP nU gnoJ miKamabO

Đặc biệt khi nối chuỗi như đọc từ File, đọc từ internet với nội dung dài thì ta nên dùng StringBuilder để nối (append) thay vì dùng dấu +. Vì khi dùng dấu + tốc độ xử lý sẽ rất chậm.

Như vậy tới đây Tui đã hướng dẫn xong một số thư viện quan trọng thường dùng trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Cần thành thạo các thư viện để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/hkqhv8emwnrep4v/HocThuVienThuongDung.rar>

Hẹn gặp các bạn ở những bài tiếp theo

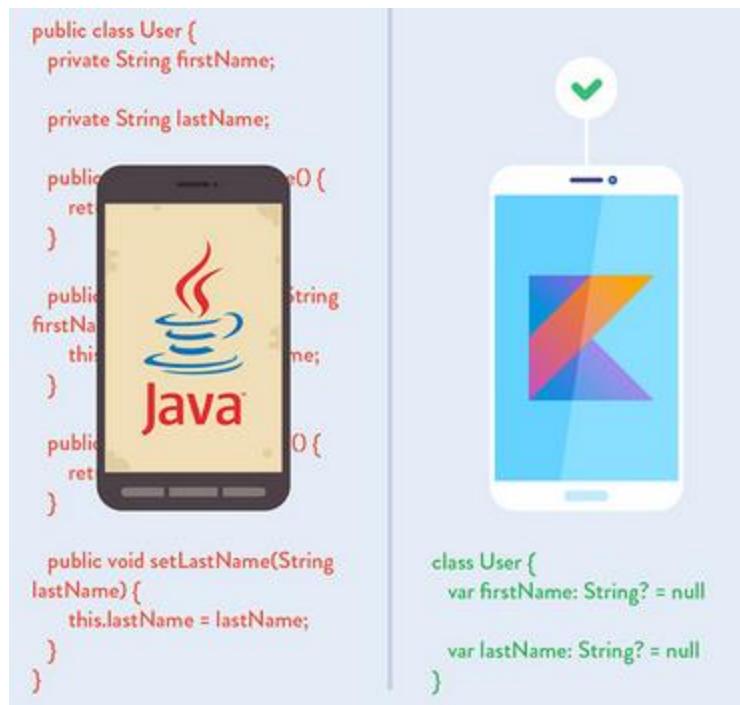
Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 18- Xử lý chuỗi trong Kotlin

Trong tất cả các ngôn ngữ lập trình thì Xử lý chuỗi vô cùng quan trọng, hầu như các dự án ta đều phải liên quan tới xử lý chuỗi.

Toàn bộ thông tin chi tiết về chuỗi trong Kotlin được trình bày tại đây <https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/>, các bạn có thời gian thì vào đó xem. Trong bài học này Tui sẽ trình bày một số hàm thường dùng nhất trong chuỗi.



Để khai báo chuỗi trong Kotlin ta dùng:

`var s:String="Obama"`

hoặc

`var s:String?= "Putin"`

Một chuỗi trong Kotlin sẽ có các thuộc tính và phương thức sau([rất rất nhiều phương thức hữu ích](#), nhưng Tui chỉ có thể liệt kê một số mà thôi):

Tên Thuộc tính/	Mô tả
-----------------	-------

phương thức	
length	thuộc tính trả về chiều dài của chuỗi
indexOf(chuỗi)	Trả về vị trí đầu tiên tìm thấy, không tìm thấy trả về -1
lastIndexOf(chuỗi)	lastIndexOf trả về vị trí cuối cùng tìm thấy
contains(chuỗi con)	Contains Kiểm tra chuỗi con có nằm trong chuỗi s?
subString(vt)	Trích lọc toàn bộ bên phải chuỗi từ vt
subString(startIndex,endIndex)	Trích lọc giữa chuỗi nằm giữa start tới end index
replace(chuỗi cũ, chuỗi mới)	Đổi toàn bộ chuỗi cũ thành chuỗi mới
replaceFirst(chuỗi cũ, chuỗi mới)	Đổi chuỗi cũ thành chuỗi mới nhưng chỉ áp dụng cho chuỗi cũ đầu tiên
trimStart	xóa khoảng trắng dư thừa bên trái
trimEnd	xóa khoảng trắng dư thừa bên phải
trim	xóa khoảng trắng dư thừa bên trái và phải
compareTo(chuỗi s2)	So sánh 2 chuỗi có phân biệt chữ HOA và chữ thường =0 khi s1=s2 >0 khi s1>s2 <0 khi s1<s2
compareTo(s2, ignoreCase = true)	So sánh 2 chuỗi KHÔNG phân biệt chữ HOA và chữ thường =0 khi s1=s2 >0 khi s1>s2 <0 khi s1<s2

plus(chuỗi x)	Nối chuỗi x vào chuỗi gốc
split(chuỗi tách)	Tách chuỗi gốc thành List<String>
toUpperCase	Chuyển chuỗi thành IN HOA
toLowerCase	Chuyển chuỗi thành in thường

Bây giờ Tui đi vào từng hàm để các bạn dễ dàng thực hành về xử lý chuỗi:

- Hàm **IndexOf** – trả về vị trí đầu tiên tìm thấy, nếu không thấy trả về -1:

```
fun main(args: Array<String>) {
    var s:String="Obama"
    var i = 0
    // Trả về i = 2
    i = s.indexOf("a")
    println(i)
}
```

Ta thấy rõ ràng ký tự/chuỗi a xuất hiện 2 lần, nhưng Kotlin chỉ quan tâm kết quả đầu tiên mà thôi, vì vậy kết quả =2 sẽ được xuất ra màn hình (vị trí bắt đầu tính từ 0).

- **lastIndexOf** trả về vị trí cuối cùng tìm thấy, nếu không thấy trả về -1:

```
fun main(args: Array<String>) {
    var s: String = "Hello everybody !";
    var i = 0;
    // Trả về i = 8
    i = s.lastIndexOf("e");
    println(i)
}
```

Ta thấy kết quả xuất ra =8 là vị trí cuối cùng của ký tự/chuỗi e

- **Contains** kiểm tra chuỗi con có tồn tại trong chuỗi gốc hay không, có thì trả về true, còn không trả về false

```
fun main(args: Array<String>) {
    var s :String = "Hello everybody !"
    var s2="body"
    if(s.contains(s2))
```

```

    {
        println("Có tồn tại [${s2}]")
    }
else
{
    println("Không tồn tại [${s2}]")
}
}

```

Kết quả chạy ta thấy xuất “Có tồn tại [body]”. Hàm contains được sử dụng rất nhiều để kiểm tra sự tồn tại của chuỗi con.

- **subString** – trích lọc chuỗi

```

fun main(args: Array<String>) {
    val s = "Xin chào Obama! Tui là Putin"
    val s2 = s.substring(9)
    println(s2)
    val s3 = s.substring(9, 14)
    println(s3)
}

```

Kết quả khi chạy ta thấy:

Obama! Tui là Putin
Obama

Muốn trích lọc bên phải chuỗi thì dùng substring với 1 đối số, muốn trích lọc giữa chuỗi thì dùng 2 đối số.

- **replace** – đổi chuỗi cũ thành chuỗi mới

```

fun main(args: Array<String>) {
    var s = "Xin chào Obama! Tui là Putin"
    s = s.replace("Obama", "Kim Jong Un")
    println(s)
}

```

Kết quả khi chạy ta thấy:

Xin chào Kim Jong Un! Tui là Putin

- **replaceFirst**-Đổi chuỗi cũ thành chuỗi mới nhưng chỉ áp dụng cho chuỗi đầu tiên tìm thấy

```
fun main(args: Array<String>) {
    var s = "Obama Xin chào Michelle Obama"
    s = s.replaceFirst("Obama", "Putin")
    println(s)
}
```

Kết quả khi chạy ta thấy:

Putin Xin chào Michelle Obama

Mặc dù chuỗi gốc ta có 2 chuỗi con Obama, nhưng chương trình chỉ áp dụng cho Obama đầu tiên

- Các hàm xóa khoảng trắng: **trim,trimEnd,trimStart**

```
fun main(args: Array<String>) {
    var s:String= "    Trần Duy Thành      "
    val s2=s.trimStart()
    println(s2+"=>size="+s2.length)
    val s3=s.trimEnd()
    println(s3+"=>size="+s3.length)
    val s4=s.trim()
    println(s4+"=>size="+s4.length)
}
```

Kết quả khi chạy ta thấy:

Trần Duy Thành =>size=20
Trần Duy Thành=>size=18
Trần Duy Thành=>size=14

- **compareTo** – so sánh chuỗi

```
fun main(args: Array<String>) {
    val s1 = "Hạnh phúc"
    val s2 = "Hạnh PHÚC"
    val x = s1.compareTo(s2, ignoreCase = true)
    println(x)
    val y = s1.compareTo(s2, ignoreCase = false)
```

```
    println(y)
}
```

Kết quả khi chạy ta thấy:

```
0
32
```

Ta thấy nếu so sánh không phân biệt chữ Hoa – thường(ignoreCase = true) thì kết quả =0, còn có phân biệt chữ HOA – thường(ignoreCase = false) thì kết quả >1 (32)

- **plus**– nối chuỗi

```
fun main(args: Array<String>) {
    var s:String="Obama"
    s=s.plus(" ")
    s=s.plus("Putin")
    println(s)
}
```

Kết quả khi chạy ta thấy:

```
Obama Putin
```

Hàm plus ở trên nối chuỗi nhưng nó không làm thay đổi chuỗi gốc mà nó trả về 1 chuỗi mới do đó ta phải lưu lại địa chỉ mới này. Ta có thể dùng dấu + (nhưng không nên dùng khi nối chuỗi quá nhiều đặc biệt đọc từ file hay internet), Ngoài ra có thể dùng StringBuilder để xử lý nối chuỗi.

- **split**– tách chuỗi theo tiêu chí bất kỳ, trả về một List<String>

```
fun main(args: Array<String>) {
    var s:String="Trần Duy Thành"
    var arr>List<String> = s.split(" ")
    println("Số phần tử="+arr.size)
    for(x in arr)
        println(x)
}
```

Kết quả khi chạy ta thấy:

```
Số phần tử=3
Trần
Duy
Thanh
```

Hàm tách chuỗi rất quan trọng, vì hầu như ta đều gặp trong quá trình viết phần mềm. Input là 1 chuỗi có quy luật, và ta phải phân tích để tách chuỗi rồi mô hình hóa thành dữ liệu có cấu trúc(thường là OOP)

- **toUpperCase, toLowerCase** để chuyển CHỮ HOA, chữ thường

```
fun main(args: Array<String>) {
    var s5:String?="Putin"
    var s:String="Trần Duy Thanh"
    var s2=s.toUpperCase()
    println(s2)
    var s3=s.toLowerCase()
    println(s3)
    println(s5)
}
```

Kết quả khi chạy ta thấy:

```
TRẦN DUY THANH
trần duy thanh
```

Bây giờ Tui thử làm một chương trình tối ưu chuỗi: Cho một chuỗi bất kỳ, hãy tối ưu chuỗi theo mô tả: Chuỗi không có khoảng trắng dư thừa, các từ cách nhau bởi 1 khoảng trắng, ký tự đầu của các từ viết HOA

Ví dụ:

Input: “ TRẦN duY THAnh ”

Output: “Trần Duy Thanh”

```
fun toiUU(s: String): String {
    var sToiuu = s
    sToiuu = sToiuu.trim()
    val arrWord = sToiuu.split(" ") ;
```

```

sToiTuu = ""
for (word in arrWord) {
    var newWord = word.toLowerCase()
    if (newWord.length > 0) {
        newWord = newWord.replaceFirst((newWord[0] + ""), 
(newWord[0] + "").toUpperCase())
        sToiTuu += newWord + " "
    }
}
return sToiTuu.trim()
}

fun main(args: Array<String>) {
    var s = " TRẦN DUY THANH "
    println(s)
    var sToiuuu = toiUU(s)
    println(sToiuuu)
    s = " NguyỄN VĂN OBAMA "
    println(s)
    sToiuuu = toiUU(s)
    println(sToiuuu)
}

```

Kết quả khi chạy ta thấy:

TRẦN	DUY	THANH
Trần	Duy	Thanh
NguyỄN	VĂN	OBAMA
Nguyễn	Văn	Obama

Như vậy tới đây Tui đã hướng dẫn xong cách xử lý chuỗi trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Cần thành thạo các thư viện để có thể áp dụng tốt vào các dự án thực tế của mình.

Cần phải đọc nhiều ví dụ về cách xử lý chuỗi từ Kotlin, vì trong bài này Tui mới liệt kê được một số phương thức thường dùng mà thôi, nó còn vô vàn các phương thức khác rất hữu ích. Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/v62acsqhh101jp8/XuLyChuoi.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 19- Xử lý mảng một chiều trong Kotlin

Với Kotlin thì Mảng là một kiểu dữ liệu rất mạnh mẽ, nó khắc phục được rất nhiều nhược điểm so với các ngôn ngữ lập trình khác như C++, C#, Java...

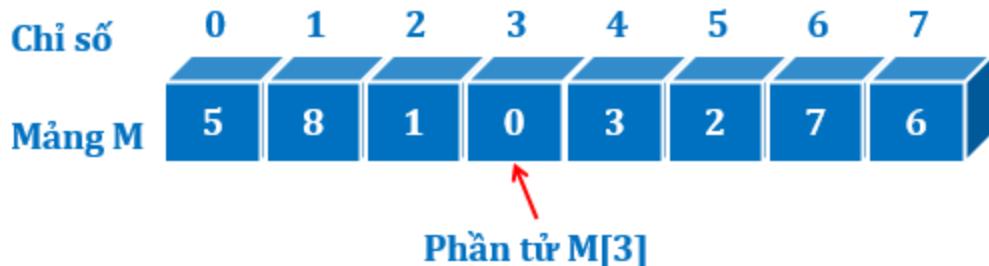
Ở [bài số 6](#) Tui đã nói sơ qua một số kiểu dữ liệu là mảng được built-in sẵn trong Kotlin. Để giúp các bạn dễ dàng hiểu được cách khai báo cũng như sử dụng mảng một chiều trong Kotlin thì Tui sẽ dùng **IntArray** để minh họa trong các ví dụ dưới đây, các kiểu mảng khác các bạn có thể tự suy luận được.

Để khai báo và cấp phát mảng ta làm như sau:

```
var M:IntArray= IntArray(n)
```

Với n là số phần tử lưu trữ tối đa của mảng.

M là một mảng lưu trữ tập các kiểu dữ liệu Int (lưu trữ tối đa n phần tử). Ta cũng có thể nói M là một đối tượng có kiểu IntArray. Tương tự như các ngôn ngữ lập trình khác, Mảng cũng lưu chỉ số từ 0, cho phép truy suất các phần tử thông qua chỉ số:



Nhưng các bạn cần chú ý là việc truy suất phân tử chỉ là một chức năng vô cùng nhỏ trong mảng của Kotlin, vì với Mảng trong Kotlin nó rất phong phú các phương thức, nó là một đối tượng được xây dựng với vô vàn phương thức xử lý rất hiệu quả: Tìm min, max, trung bình, tổng, tìm kiếm, sắp xếp...

Tui liệt kê một số thuộc tính và phương thức thường dùng của Mảng (dĩ nhiên còn rất nhiều phương thức khác, khi nào gặp thì các bạn nghiên cứu thêm):

Tên Thuộc tính/phương thức	Mô tả
size	Thuộc tính trả về kích thước thực sự của mảng

[i]	Indexer cho phép truy suất và thay đổi giá trị tại vị trí i của mảng
count/count{ }	đếm/ đếm có điều kiện
min()	hàm trả về số nhỏ nhất trong mảng
max()	hàm trả về số lớn nhất trong mảng
sum()	hàm trả về tổng mảng
average()	hàm trả về trung bình mảng
sort()	sắp xếp mảng tăng dần
sortDescending()	sắp xếp mảng giảm dần
filter{ }	Tìm kiếm/ lọc danh sách trong mảng
reverse()	Đảo mảng
contains()	Kiểm tra Mảng có chứa phần tử nào đó hay không

Ví dụ khai báo mảng M có khả năng chứa tối đa 10 phần tử:

var M:IntArray= IntArray(10)

Nhập giá trị cho từng phần tử trong mảng M:

M[0]=100

M[1]=20

...

...

M[9]=-5

Vì mảng M có khả năng chứa 10 phần tử nên các Indexer sẽ chạy từ 0 → 9

Để lấy giá trị tại một vị trí bất kỳ:

```
var x:Int=M[2]
```

Để duyệt các phần tử trong mảng ta có thể duyệt theo vị trí như sau:

```
for (i in M.indices)  
print("${M[i]}\t")
```

hoặc duyệt giá trị theo cách:

```
for (i in M)  
print("$i\t")
```

Các hàm khác trong mảng cũng dễ dàng thực hiện.

Ví dụ chi tiết tạo mảng M có 10 phần tử với các giá trị ngẫu nhiên:

```
import java.util.*  
  
/**  
 * Created by cafe on 28/05/2017.  
 */  
fun main(args: Array<String>) {  
    var M:IntArray= IntArray(10)  
  
    var rd=Random()  
    for(i in M.indices)  
        M[i]=rd.nextInt(100)  
    println("Mảng sau khi nhập - duyệt theo giá trị:")  
    for (i in M)  
        print("$i\t")  
    println()  
    println("Mảng sau khi nhập - duyệt theo vị trí:")  
    for (i in M.indices)  
        print("${M[i]}\t")  
    println()  
    //số lớn nhất  
    println("MAX=${M.max()}"")  
    //số nhỏ nhất  
    println("MIN=${M.min()}"")  
    //tổng mảng  
    println("SUM=${M.sum()}"")  
    //trung bình mảng  
    println("AVERAGE=${M.average()}"")  
    //đếm số chẵn  
    println("Số chẵn=${M.count { x->x%2==0 }}")
```

```

//đếm số lẻ
println("Số lẻ=${M.count { x->x%2==1 }}")
//sắp xếp tăng dần
M.sort()
println("Tăng dần:")
for (i in M)
    print("$i\t")
println()
//sắp xếp giảm dần
M.sortDescending()
println("Giảm dần:")
for (i in M)
    print("$i\t")
println()
//lọc các số chẵn trong mảng
var dsChan= M.filter { x->x%2==0 }
println("Các số chẵn:")
for (i in dsChan)
    print("$i\t")
println()
//lọc các số lẻ trong mảng
var dsLe= M.filter { x->x%2==1 }
println("Các số Lẻ:")
for (i in dsLe)
    print("$i\t")
println()
var k=50
//lọc các số >50 trong mảng
var dsTim=M.filter { x->x>k }
println("Các số > $k:")
for (i in dsTim)
    print("$i\t")
println()
}

```

Kết quả khi chạy ta thấy:

Mảng sau khi nhập – duyệt theo giá trị:

47 50 51 71 63 96 65 91 1 90

Mảng sau khi nhập – duyệt theo vị trí:

47 50 51 71 63 96 65 91 1 90

MAX=96

MIN=1

SUM=625

AVERAGE=62.5

Số chẵn=3

Số lẻ=7

Tăng dần:

1	47	50	51	63	65	71	90	91	96
---	----	----	----	----	----	----	----	----	----

Giảm dần:

96	91	90	71	65	63	51	50	47	1
----	----	----	----	----	----	----	----	----	---

Các số chẵn:

96	90	50
----	----	----

Các số Lẻ:

91	71	65	63	51	47	1
----	----	----	----	----	----	---

Các số > 50:

96	91	90	71	65	63	51
----	----	----	----	----	----	----

Như vậy tới đây Tui đã hướng dẫn xong cách xử mảng 1 chiều trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Cần thành thạo các thư viện để có thể áp dụng tốt vào các dự án thực tế của mình.

Bài sau Tui sẽ trình bày về mảng 2 chiều trong Kotlin

Các bạn có thể tải source code bài này ở đây:

http://www.mediafire.com/file/yha9wwfu1879ac4/HocXuLyMang_1d.rar

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 20- Xử lý mảng hai chiều trong Kotlin

Ở [bài 19](#) chúng ta đã thao tác được với mảng 1 chiều. Trong bài này Tui sẽ trình bày về cách khai báo và sử dụng mảng 2 chiều của Kotlin.

Cú pháp khai báo mảng 2 chiều tổng quát trong Kotlin:

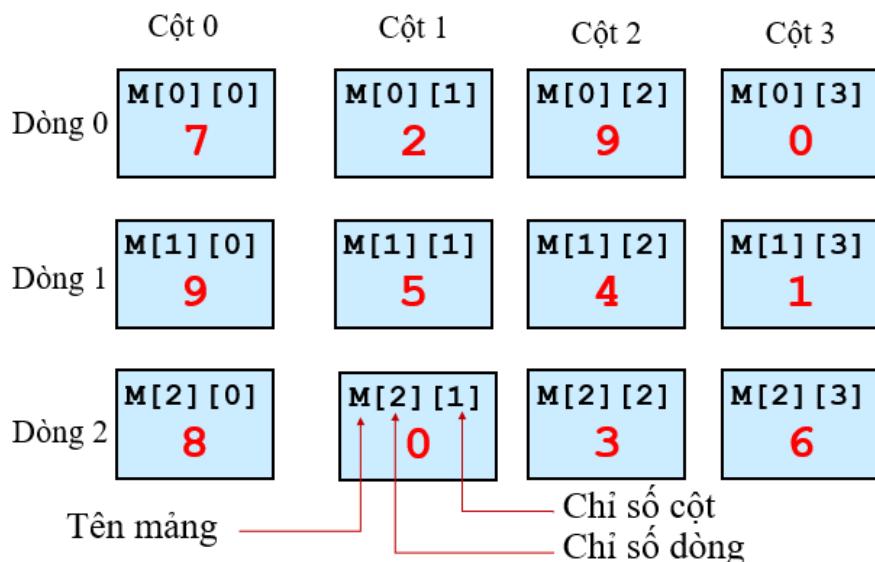
```
var M:Array<Kiểu_Dữ_Liệu_Mảng> = Array(Số_Dòng,{Kiểu_Dữ_Liệu_Mảng(Số_Cột)})
```

Ví dụ dưới đây sẽ khai báo một mảng 2 chiều tên là M có 10 dòng và 5 cột, dữ liệu các phần tử trong M có kiểu Int:

```
var M:Array<IntArray> = Array(10,{IntArray(5)})
```

Các chỉ số dòng và cột cũng chạy từ 0->n-1

Bạn có thể tưởng tượng cấu trúc dữ liệu của mảng 2 chiều như sau:



Mảng 2 chiều trong Kotlin cũng có một tập các phương thức vô cùng mạnh mẽ.

Để nhập liệu cho các phần tử trong mảng 2 chiều M ta làm như sau:

```
var rd:Random = Random()
for(i in M.indices)
{
    for(j in M[i].indices)
```

```

{
M[i][j]=rd.nextInt(100)
}
}

```

Ta dùng 2 vòng lặp for duyệt theo vị trí (indices). Vòng lặp ngoài là duyệt theo dòng, vòng lặp trong là duyệt theo cột. Mỗi lần duyệt ta có $M[i][j]$ chính là phần tử ở dòng thứ i và cột thứ j. Ta cập nhật dữ liệu cho phần tử này.

Để xuất dữ liệu ra màn hình ta cũng làm tương tự:

```

for(i in M.indices)
{
for(j in M[i].indices)
{
print("${M[i][j]}\t")
}
println()
}

```

Ví dụ chi tiết một số chức năng xử lý Mảng 2 chiều trong Kotlin:

```

import java.util.*

/**
 * Created by cafe on 28/05/2017.
 */
fun main(args: Array<String>) {
    var M:Array<IntArray> = Array(10, { IntArray(5) })
    var rd:Random = Random()
    for(i in M.indices)
    {
        for(j in M[i].indices)
        {
            M[i][j]=rd.nextInt(100)
        }
    }
    println("Mảng 2 chiều sau khi nhập:")
    for(i in M.indices)
    {
        for(j in M[i].indices)
        {
            print("${M[i][j]}\t")
        }
        println()
    }
}

```

```

    }
    println("Mảng 2 chiều sau khi nhập - cách 2:")

    for (row in M)
    {
        for (cell in row)
        {
            print("$cell \t")
        }
        println()
    }
    println("Mảng thứ 1:")
var M1=M[1]
for (i in M1.indices)
    print("${M1[i]}\t")
    println()
}

```

Kết quả khi chạy ta thấy:

Mảng 2 chiều sau khi nhập:				
51	9	58	96	46
48	76	11	68	44
11	73	77	18	45
22	10	91	22	35
59	66	48	42	59
84	5	58	0	79
50	68	72	29	64
8	66	69	90	93
35	92	5	8	13
18	48	44	82	52
Mảng 2 chiều sau khi nhập - cách 2:				
51	9	58	96	46
48	76	11	68	44
11	73	77	18	45
22	10	91	22	35
59	66	48	42	59
84	5	58	0	79
50	68	72	29	64
8	66	69	90	93
35	92	5	8	13
18	48	44	82	52
Mảng thứ 1:				
48	76	11	68	44

Như vậy tới đây Tui đã hướng dẫn xong cách khai báo, nhập, xuất mảng 2 chiều trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Các bạn Cần kiểm tra thêm nhiều phương thức khác trong mảng 2 chiều để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

http://www.mediafire.com/file/sxi7swegsgcsz17/HocXuLyMang_2d.rar

Hẹn gặp các bạn ở những bài tiếp theo

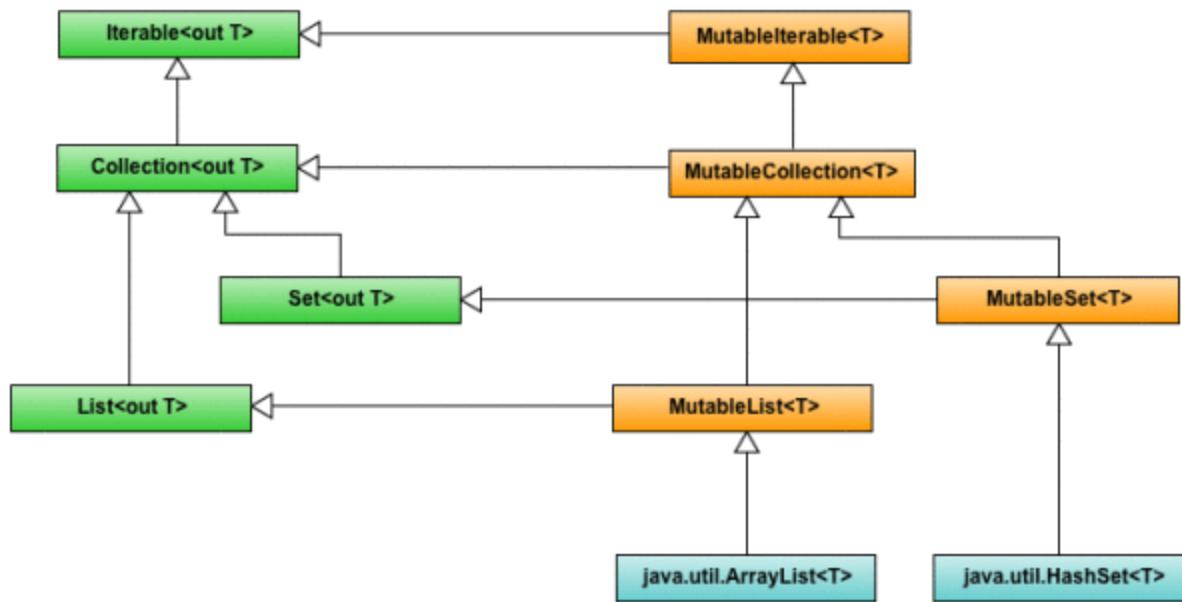
Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 21-Collections trong Kotlin

Không giống như các ngôn ngữ lập trình khác, Kotlin phân biệt rõ 2 loại Collections(Mutable collections và Immutable collections).

Mô hình lớp kế thừa của các Collection trong Kotlin/java:



Mutable Collections là tập các lớp dùng để lưu trữ danh sách dữ liệu và **có thể** thay đổi kích thước được

Immutable Collections là tập các lớp dùng để lưu trữ danh sách dữ liệu và **không thể** thay đổi kích thước được

Cả 2 loại Collections này rất dễ tạo và sử dụng, chỉ khác nhau chút xíu ở mục đích sử dụng của lập trình viên.

Trong giới hạn bài học này Tui chỉ trình bày về **MutableList** và **List**, các lớp Collection khác các bạn tự tìm hiểu thêm nếu trong dự án có gặp. Tuy nhiên với **MutableList** và **List** thì theo Tui bạn đã có thể xử lý được hầu hết các trường hợp lưu trữ, tương tác, hiển thị dữ liệu trong phần mềm rồi.

MutableList Là collection có thể thay đổi kích thước dữ liệu: Có thể thêm, sửa, xóa...

List Là collection chỉ có nhiệm vụ readOnly, dùng để hiển thị thông tin. Và dĩ nhiên nó sẽ tối ưu bộ nhớ hơn so với **MutableList**. Do đó nếu như bạn chỉ muốn hiển thị thông tin thì nên dùng **List**.

Các collection trong Kotlin không có các Constructor khởi tạo riêng, mà nó thông qua các hàm **mutableListOf()**, **listOf()** để khởi tạo

Ví dụ 1 dưới đây minh họa cách khởi tạo 2 collections:

```
var ds1:MutableList<Int> = mutableListOf()  
var ds2>List<Int> = listOf()
```

Ở trên 2 đối tượng ds1 và ds2 được khởi tạo với danh sách rỗng.

Ta có thể khởi tạo với một số dữ liệu ban đầu,

Ví dụ 2

```
var ds1:MutableList<Int> = mutableListOf(5, 6, 1, 0, 4)  
var ds2>List<Int> = listOf(1, 2, 3, 4)
```

Ở trên ds1 được khởi tạo mặc định với 5 phần tử và ta có thể thay đổi thông tin, kích thước ds1

ds2 được khởi tạo mặc định với 4 phần tử và ta không thể thay đổi thông tin, kích thước ds2. Khi ta dùng List tức là trong đầu ta muốn rằng nó là readOnly, chỉ hiển thị dữ liệu mà thôi.

Dưới đây là một số phương thức thường dùng với MutableList/List:

Tên Thuộc tính/phương thức	Mô tả
size	Thuộc tính trả về kích thước thực sự của Collection
[i]	Indexer cho phép truy suất và thay đổi giá trị tại vị trí i của collection
add()	Thêm một phần tử
addAll()	Thêm nhiều phần tử

removeAt()	Xóa theo vị trí
remove()	Xóa theo đối tượng
removeIf{ }	Xóa theo điều kiện
clear()	Xóa toàn bộ danh sách
sort()	Sắp xếp tăng dần
sortDescending()	Sắp xếp giảm dần
filter { }	Lọc dữ liệu
contains()	Kiểm tra Collection có chứa phần tử nào đó hay không

Và còn nhiều các phương thức lợi hại khác, các bạn tự tìm hiểu thêm nhé.

Ví dụ 3 – Thao tác với List:

```
fun main(args: Array<String>) {
    var ds:List<Int> = listOf(5, 6, 1, 9, -4, 7, 8, 2)
    println("Các phần tử trong List cách 1:")
    for(i in ds.indices)
        print("${ds[i]}\t")
    println()
    println("Các phần tử trong List cách 2:")
    for(i in ds)
        print("$i\t")
    println()
    println("Các phần tử sắp xếp tăng dần:")
    var ds2= ds.sorted()
    for(i in ds2)
        print("${i}\t")
    println()
    println("Các phần tử sắp xếp giảm dần:")
    var ds3= ds.sortedDescending()
    for(i in ds3)
        print("${i}\t")
    println()
    var ds4=ds.filter { x->x%2==0 }
    println("Các phần tử chẵn:")
    for(i in ds4)
```

```

        print("$i\t")
    println()
    println("SUM=" + ds.sum())
    println("MAX=" + ds.max())
    println("MIN=" + ds.min())
    println("AVERAGE=" + ds.average())
}

```

Kết quả khi chạy ta thấy:

```

Các phần tử trong List cách 1:
5   6   1   9   -4   7   8   2
Các phần tử trong List cách 2:
5   6   1   9   -4   7   8   2
Các phần tử sắp xếp tăng dần:
-4   1   2   5   6   7   8   9
Các phần tử sắp xếp giảm dần:
9   8   7   6   5   2   1   -4
Các phần tử chẵn:
6   -4   8   2
SUM=34
MAX=9
MIN=-4
AVERAGE=4.25

```

Ta để ý các phương thức của List đa phần không làm thay đổi nội tại List mà nó trả về một List mới.

Ví dụ 4 – Thao tác với MutableList:

```

fun main(args: Array<String>) {
    var ds:MutableList<Int> = mutableListOf()
    ds.add(10) //thêm một phần tử có giá trị 10
    ds.add(4) //thêm một phần tử có giá trị 4
    ds.add(5) //thêm một phần tử có giá trị 5
    ds.add(8) //thêm một phần tử có giá trị 8
    ds.addAll(mutableListOf(9,0,7)) //thêm một danh sách có 3
phần tử: 9,0,7
    println("Các phần tử trong MutableList - theo giá trị:")
    for(i in ds)
        print("$i\t")
    println()
    println("Các phần tử trong MutableList - theo vị trí:")
    for(i in ds.indices)
        print("${ds[i]}\t")
}

```

```

println()
ds[2]=113//đổi giá trị phần tử thứ 2 thành 113
println("Các phần tử trong MutableList sau khi đổi:")
for(i in ds.indices)
    print("${ds[i]}\t")
println()
ds.removeAt(3)//xóa phần tử tại vị trí thứ 3
println("Các phần tử trong MutableList sau khi xóa:")
for(i in ds.indices)
    print("${ds[i]}\t")
println()
ds.sort()//sắp tăng dần
println("Các phần tử trong MutableList sau sắp tăng dần:")
for(i in ds.indices)
    print("${ds[i]}\t")
println()
ds.sortDescending()//sắp giảm dần
println("Các phần tử trong MutableList sau sắp giảm dần:")
for(i in ds.indices)
    print("${ds[i]}\t")
println()
println("SUM="+ds.sum())
println("MAX="+ds.max())
println("MIN="+ds.min())
println("AVERAGE="+ds.average())
//lọc các số lẻ
var dsLe=ds.filter { x->x%2==1 }
println("Các phần tử trong MutableList là số lẻ:")
for(i in dsLe.indices)
    print("${dsLe[i]}\t")
println()
var dsNT=ds.filter {
    x->
        var dem=0
        for(i in 1..x)
        {
            if(x%i==0)
                dem++
        }
        dem==2
}
println("Các phần tử trong MutableList là số nguyên tố:")
for(i in dsNT.indices)
    print("${dsNT[i]}\t")
println()
}

```

Kết quả khi chạy ta thấy:

```
Các phần tử trong MutableList - theo giá trị:  
10      4      5      8      9      0      7  
Các phần tử trong MutableList - theo vị trí:  
10      4      5      8      9      0      7  
Các phần tử trong MutableList sau khi đổi:  
10      4      113     8      9      0      7  
Các phần tử trong MutableList sau khi xóa:  
10      4      113     9      0      7  
Các phần tử trong MutableList sau sắp tăng dần:  
0      4      7      9      10     113  
Các phần tử trong MutableList sau sắp giảm dần:  
113     10     9      7      4      0  
SUM=143  
MAX=113  
MIN=0  
AVERAGE=23.83333333333332  
Các phần tử trong MutableList là số lẻ:  
113     9      7  
Các phần tử trong MutableList là số nguyên tố:  
113     7
```

Ta thấy collection trong Kotlin rất linh động và mạnh mẽ, ta có thể tùy biến trong vấn đề xử lý. Hàm filter như là một cuộc cách mạng trong lọc dữ liệu.

Như vậy tới đây Tui đã hướng dẫn xong cách xử 2 Collection quan trọng trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Cần thành thạo các Collection để có thể áp dụng tốt vào các dự án thực tế của mình.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/d1b3imc09aw60et/HocXuLyCollection.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 22-Lập trình hướng đối tượng trong Kotlin – phần 1

Kotlin giống như các ngôn ngữ lập trình khác đó là hỗ trợ cài đặt lập trình hướng đối tượng. Và các bạn cần chú ý rằng, Lập trình hướng đối tượng là một khái niệm chung, ngôn ngữ lập trình chỉ là một trong những công cụ để triển khai khái niệm đó mà thôi. Điều này có nghĩa là nếu bạn đã hiểu OOP(object oriented programming) rồi thì C++, C#, Java, PHP, Python, Kotlin, R, GO....hay bất kỳ một ngôn ngữ nào khác sẽ dùng chung khái niệm OOP này để cài đặt, chỉ là kỹ thuật xử lý OOP của các ngôn ngữ này nó khác nhau.

Bài này Tui sẽ đi qua một số khái niệm liên quan tới OOP, sau đó ta sẽ làm quen với một ví dụ cài đặt OOP bằng Kotlin. Những khái niệm về OOP nó dày rẫy trên mạng, các bạn có thể tự search và đọc thêm các chi tiết khác.

Một số đặc điểm của OOP:

- Tập trung vào dữ liệu thay cho các hàm.
- Chương trình được chia thành các đối tượng độc lập.
- Cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng.
- Dữ liệu được che giấu, bao bọc.
- Các đối tượng trao đổi với nhau thông qua các hàm.
- Chương trình được thiết kế theo hướng tiếp cận từ dưới lên (bottom-up): *Phương pháp xây dựng chương trình bottom-up là phương pháp thường dùng để xây dựng phần mềm lớn, phức tạp. Ý tưởng của phương pháp này là xuất phát từ nhiều thành phần nhỏ đã có sẵn, ta khéo kết hợp chúng lại để tạo ra thành phần chức năng lớn hơn, ta tiếp tục kết hợp các thành phần xây dựng được để tạo ra thành phần lớn hơn nữa... cho đến khi xây dựng được chương trình giải quyết được bài toán mong muốn.*

Một số ưu điểm nổi bật của OOP:

- Không có nguy cơ dữ liệu bị thay đổi tự do trong chương trình.
- Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi mã nguồn của các đối tượng khác.
- Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên.
- Phù hợp với các dự án phần mềm lớn, phức tạp.

Khái niệm **đối tượng** (object) trong lập trình hướng đối tượng giống như một đối tượng cụ thể trong thế giới thực.

Mỗi đối tượng có các thuộc tính và các hành vi riêng:

- **Thuộc tính** (attribute) mô tả đặc điểm của đối tượng.
- Hành vi là phương thức hoạt động của đối tượng, gọi tắt là **phương thức** (method).

Ví dụ 1: Đối tượng Phân số

Đặc điểm của Phân số (thuộc tính) có:

- Tử số
- Mẫu số

Các Thao tác trên phân số (hành vi)

- Cộng, trừ, nhân, chia
- Tối giản
- Nghịch đảo

Ví dụ 2: Đối tượng Xe hơi

Đặc điểm của Xe hơi (thuộc tính) có:

- Hiệu xe
- Màu xe
- Số bánh xe
- Số cửa

Các Thao tác trên Xe hơi (hành vi):

- Chạy tới
- Chạy lui
- Dừng xe

Các đối tượng có các đặc điểm (thuộc tính và phương thức) giống nhau được gom nhóm thành một lớp để phân biệt với các đối tượng khác và dễ quản lý.

Một lớp (class) là sự phân loại của các đối tượng hay là kiểu (type) của đối tượng.

Ví dụ 3:

– Các chiếc xe Toyota, Honda, Porsche thuộc lớp xe hơi.

– Các con chó giữ nhà, chó săn, chó kiêng thuộc lớp chó.

-Bia Ken, Tiger, Sài gòn đỗ.. được gom lại thành một lớp Bia chǎng hạn

Như vậy **Lớp(Class)** là một khái niệm trừu tượng, dùng để chỉ một tập hợp các đối tượng có mặt trong hệ thống.

Lớp có thuộc tính và phương thức:

- Thuộc tính của lớp tương ứng với thuộc tính của đối tượng.
- Phương thức của lớp tương ứng với các hành động của đối tượng.

Một **Lớp** có thể có một trong các khả năng sau:

- Không có thuộc tính, không có phương thức.
- Hoặc chỉ có thuộc tính, không có phương thức.
- Hoặc chỉ có phương thức, không có thuộc tính.
- Hoặc có cả thuộc tính và phương thức, trường hợp này là phổ biến nhất.
- Có các mối quan hệ với các lớp khác

Gói (package)

Một nhóm các lớp (classes) và giao diện (interfaces) được tổ chức thành một đơn vị quản lý theo hình thức không gian tên gọi là package.

Lợi ích của package là tổ chức sắp xếp lại hệ thống thông tin các lớp trong dự án một cách khoa học, giúp cho việc theo dõi bảo trì dự án được tốt nhất.

Tính trừu tượng:

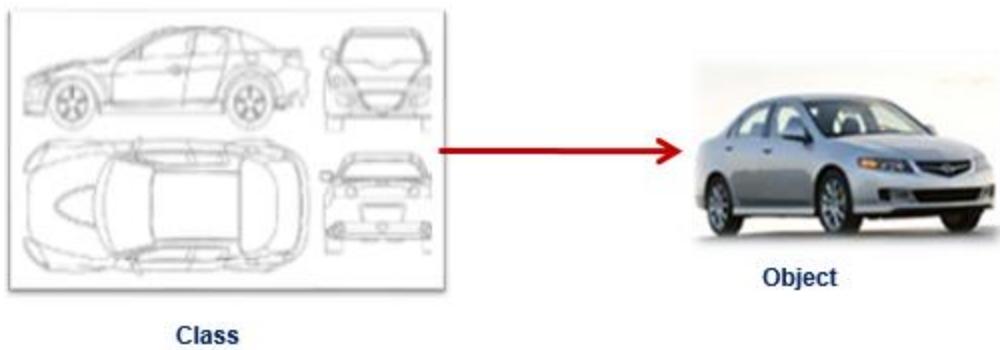
Trường hợp 1:

Lớp (Class) là một khái niệm trừu tượng, đối tượng là một thể hiện cụ thể của lớp.

Ví dụ 4:

Bản thiết kế của chiếc xe hơi là lớp.

Chiếc xe hơi được tạo ra từ bản thiết kế là đối tượng.



Trường hợp 2:

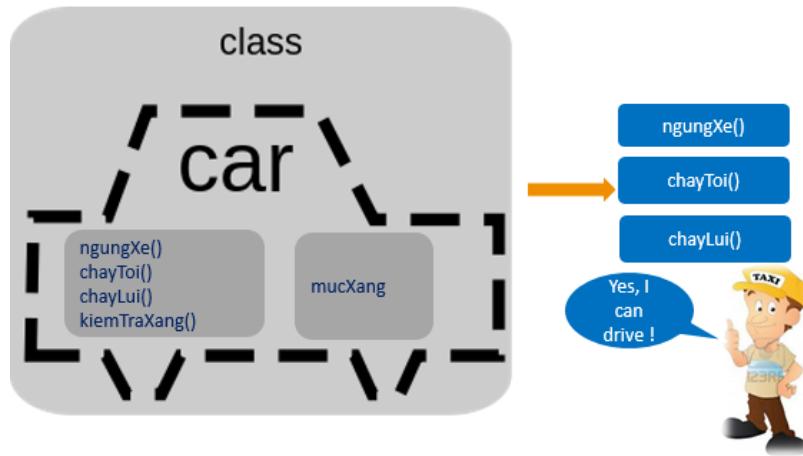
Từ những đối tượng giống nhau: trừu tượng hóa thành một lớp: Chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.



Tui nghĩ bạn nào cũng học Sinh Học, bạn thấy trong Sách giáo khoa người ta cũng phân ra : Lớp Chim, Lớp động vật có vú, lớp động vật ăn cỏ.... đó chính là sự trừu tượng hóa, tự tập các đối tượng họ trừu tượng thành các lớp.

Tính đóng gói:

- Mỗi lớp được xây dựng để thực hiện một nhóm chức năng đặc trưng của riêng lớp đó.
- Tất cả mọi thao tác truy xuất vào thành phần dữ liệu từ đối tượng này qua đối tượng khác phải được thực hiện bởi các phương thức (method) của chính đối tượng chứa dữ liệu.
- Tính đóng gói cho phép dấu thông tin của đối tượng bằng cách kết hợp thông tin và các phương thức liên quan đến thông tin trong đối tượng.



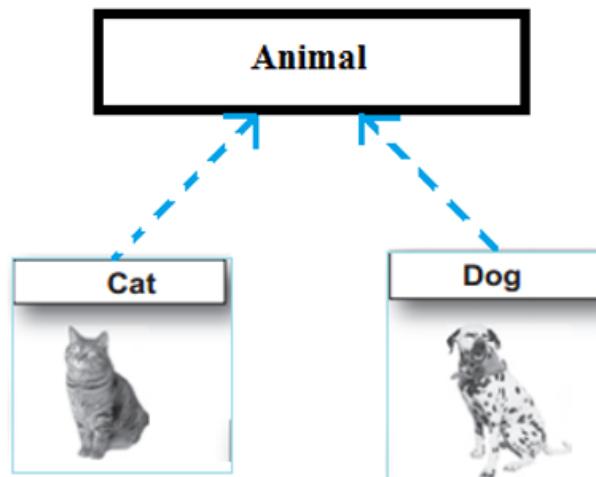
Tính kế thừa:

- Cho phép xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có.
- Lớp đã có gọi là lớp Cha, lớp mới phát sinh gọi là lớp Con
- Lớp con kế thừa tất cả các thành phần của lớp Cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.

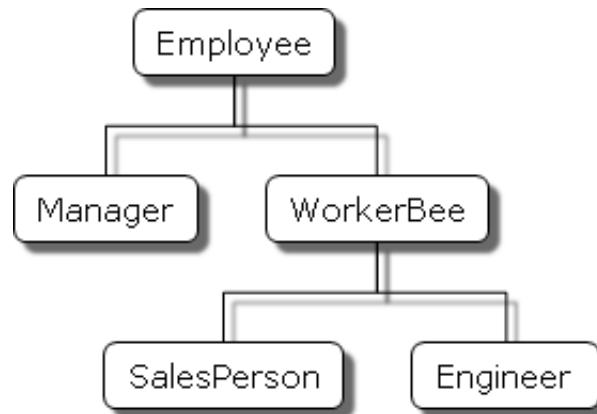
ví dụ về sự phát triển các dòng xe theo thời gian:



Ví dụ về kế thừa:

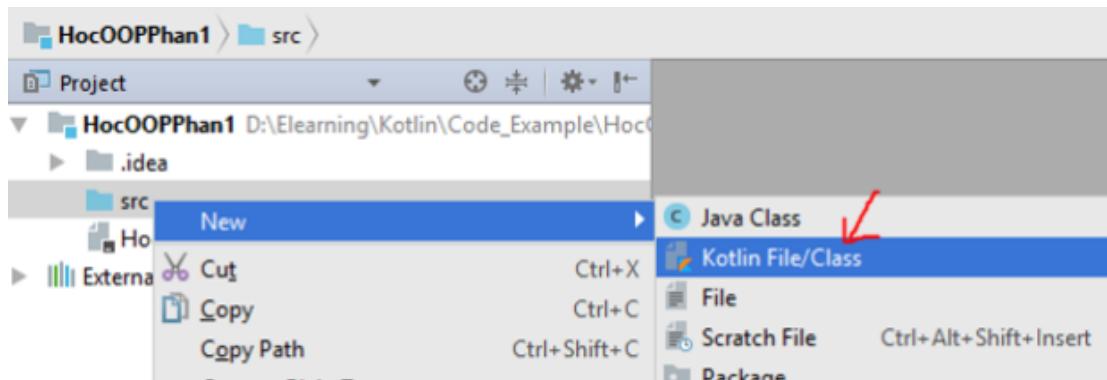


Ví dụ về kế thừa trong Employee:

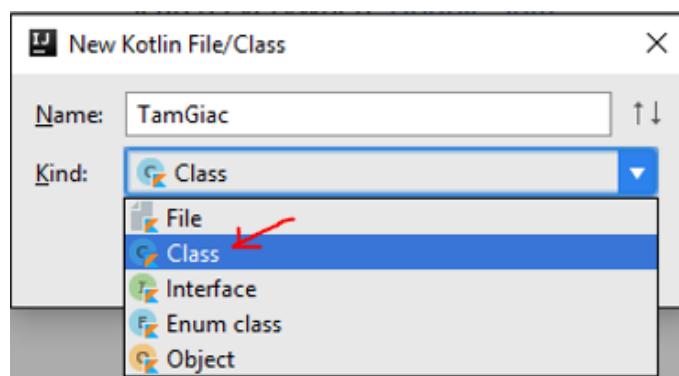


Dưới đây là ví dụ về tạo một Lớp trong Kotlin (trong bài này Tui sẽ không giải thích chi tiết các thành phần trong Kotlin, các bài sau Tui sẽ trình bày rõ):

Ta tạo một Project Kotlin với tên HocOOPPhan1 rồi thêm 1 Class Kotlin theo hình dưới đây:

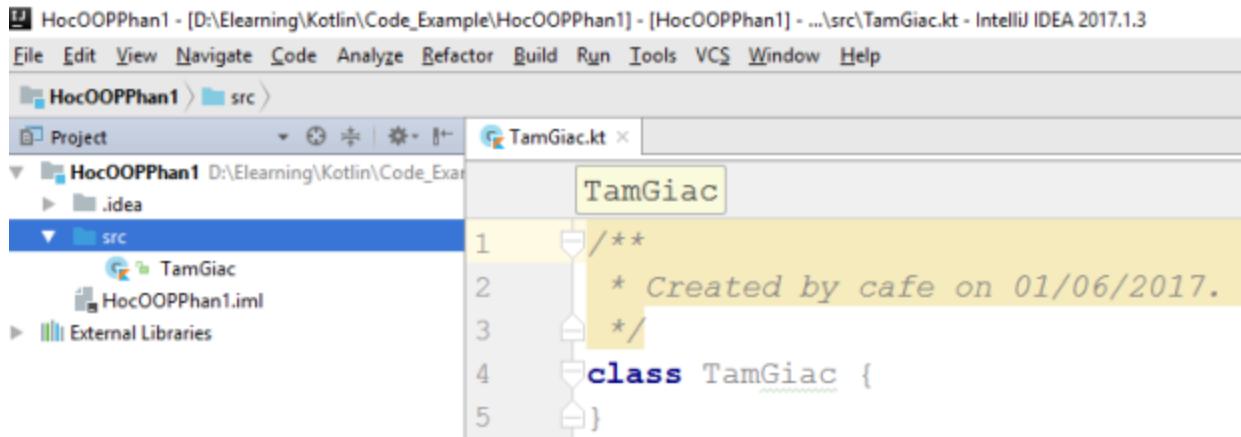


Bấm chuột phải vào src/ chọn Kotlin File/Class:



Ở màn hình New Kotlin File/Class ta chọn mục Kind là Class, đặt tên lớp là TamGiac
→ bấm OK để tạo

Kết quả đầu tiên khi tạo 1 Lớp trong Kotlin:



The screenshot shows the IntelliJ IDEA interface with the title bar "HocOOPPhan1 - [D:\Elearning\Kotlin\Code_Example\HocOOPPhan1] - [HocOOPPhan1] - ...\\src\\TamGiac.kt - IntelliJ IDEA 2017.1.3". The menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help. The left sidebar shows the project structure with "HocOOPPhan1" and ".idea" at the top, followed by "SRC" which contains "TamGiac" and "HocOOPPhan1.iml". The right panel displays the code editor with the file "TamGiac.kt" open. The code is as follows:

```
1 /**
2 * Created by cafe on 01/06/2017.
3 */
4 class TamGiac {
```

Tiến hành code cho TamGiac (bạn cứ làm theo, Tui sẽ giải thích kỹ các thành phần ở bài sau):

Chi tiết lớp TamGiac:

```
class TamGiac {
    var a:Double=0.0
    var b:Double=0.0
    var c:Double=0.0
    constructor()
    constructor(a:Double,b:Double,c:Double)
    {
        this.a=a
        this.b=b
        this.c=c
    }
    fun chuVi():Double
    {
        return a+b+c
    }
    fun dienTich():Double
    {
        var p=chuVi()/2
        return Math.sqrt(p*(p-a)*(p-b)*(p-c))
    }
}
```

Tiếp tục tạo 1 file Kotlin app_test_tamgiac.kt để sử dụng TamGiac:

```
fun main(args: Array<String>) {
    var tg1=TamGiac(4.0,5.0,6.0)
    println("Thông tin tam giác 1:")
    println("Chu vi="+tg1.chuVi())
    println("Diện Tích="+tg1.dienTich())

    var tg2=TamGiac()
    tg2.a=7.5
    tg2.b=10.3
    tg2.c=15.5
    println("Thông tin tam giác 2:")
    println("Chu vi="+tg2.chuVi())
    println("Diện Tích="+tg2.dienTich())
}
```

Kết quả khi chạy ta thấy:

```
Thông tin tam giác 1:
Chu vi=15.0
Diện Tích=9.921567416492215
Thông tin tam giác 2:
Chu vi=33.3
Diện Tích=33.354424275499014
```

Như vậy tới đây Tui đã trình bày xong một số khái niệm về OPP cũng như cách tạo một Lớp và sử dụng nó như thế nào trong Kotlin, các bạn chú ý học kỹ và hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ trình bày chi tiết từng thành phần của Class trong Kotlin

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/1ms2t5jdfjxo94k/HocOOPPhan1.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 23-Lập trình hướng đối tượng trong Kotlin – phần 2

Trong [bài 22](#) Tui đã điểm qua một số đặc điểm của OOP cùng với một ví dụ về tạo Lớp và sử dụng Lớp trong Kotlin. Trong bài này Tui sẽ đi vào giải thích chi tiết từng thành phần trong quá trình tạo Lớp.

Cấu trúc của một **class** thường có:

- Tên Lớp
- Các constructors
- Các thuộc tính
- Các getter/setter
- Các phương thức

1) Khai báo Tên Lớp trong Kotlin:

Với Kotlin, để tạo một lớp ta dùng từ khóa **class** giống như các ngôn ngữ lập trình khác:

```
class SinhVien {  
}
```

Không giống với các ngôn ngữ khác, Kotlin cho phép khai báo chỉ mỗi Tên Lớp mà không cần có bất kỳ thành phần nào khác, cách khai báo này như sau:

```
class SinhVien
```

Tên lớp ta nên đặt theo quy tắc: Ký Tự Đầu Tiên của các Từ phải Viết Hoa, không chứa khoảng trắng...

Kotlin hỗ trợ 2 loại Constructors (những hàm đặc biệt, mặc định để khởi tạo các giá trị ban đầu cho các đối tượng khi cấp phát bộ nhớ) đó là **primary constructor** và **secondary constructors**

2) Khai báo primary constructor trong Kotlin:

```
class SinhVien constructor(ma:Int,ten:String) {  
}
```

Ở trên constructor là từ khóa, bên trong nó có 2 đối số, có nghĩa là Lớp SinhVien này có 1 Primary constructor có 2 đối số.

Nếu như primary constructor không chứa các chú thích (annotations) và các modifiers(private, protected,public) thì có thể loại bỏ từ khóa constructor khỏi khai báo, tức là ta có thể Khai báo ngắn gọn như sau:

```
class SinhVien (ma:Int, ten:String){  
}
```

Một lưu ý quan trọng là primary constructor không thể chứa bất kỳ đoạn mã nào, nếu muốn khởi tạo các giá trị mặc định cho các biến khi dùng primary constructor thì ta phải dùng khối lệnh **init{}**, ví dụ:

```
class SinhVien (ma:Int, ten:String){  
    init {  
        println("Đây là primary constructor")  
        println("Mã=$ma ; Tên =$ten")  
    }  
}
```

Khi khai báo biến để sử dụng lớp SinhVien ở trên ta sẽ làm như sau:

```
fun main(args: Array<String>) {  
    var lanh=SinhVien(113,"Trần Thị Long Lanh")  
}
```

Chạy chương trình, ta được kết quả như sau:

```
Đây là primary constructor  
Mã=113 ; Tên =Trần Thị Long Lanh
```

Khi khai báo **var lanh=SinhVien(113,"Trần Thị Long Lanh")** nó sẽ tự động tìm tới primary constructor và tự nhảy vào init block để thực hiện.

Và chú ý là **không cần dùng từ khóa new** để xin cấp phát bộ nhớ như C# hay java...

3)Khai báo secondary constructor trong Kotlin:

Ta cũng dùng từ khóa constructor để khai báo secondary constructor trong kotlin, ví dụ:

```

class SinhVien {
    constructor() {
        println("Đây là secondary constructor 0 đối số")
    }
    constructor(ma:Int,ten:String) {
        println("Đây là secondary constructor 2 đối số")
        println("Mã=$ma ; Tên =$ten")
    }
}

```

Khai báo sử dụng các secondary constructor:

```

fun main(args: Array<String>) {
    var teo=SinhVien()
    var lanh=SinhVien(113,"Trần Thị Long Lanh")
}

```

Khi chạy phần mềm ta có các thông báo sau:

```

Đây là secondary constructor 0 đối số
Đây là secondary constructor 2 đối số
Mã=113 ; Tên =Trần Thị Long Lanh

```

4) Khai báo các thuộc tính của Lớp trong Kotlin:

Thuộc tính là những gì thuộc về đối tượng, ví dụ như 1 Sinh Viên có các thông tin: mã, tên, năm sinh... thì các thông tin này là các thuộc tính của đối tượng Sinh Viên

Kotlin giống như các ngôn ngữ khác, cung cấp một số các Visibily Modifiers (private, protected, public, default) cho các thuộc tính:

Modifiers	Lớp (Class)	Gói (Package)	Lớp con (Subclass)	Ngoài
public	Có	Có	Có	Có
protected	Có	Có	Có	Không

Không có (no modifier – default)	Có	Có	có	Không
private	Có	Không	Không	Không

Ví dụ:

```
class SinhVien {
    var ma:Int=0
    var ten:String=""
    constructor()
    {
        println("Đây là secondary constructor 0 đối số")
    }
    constructor(ma:Int,ten:String)
    {
        println("Đây là secondary constructor 2 đối số")
        println("Mã=$ma ; Tên =$ten")
    }
}
```

Theo khai báo ở trên thì 2 thuộc tính ma và ten đều là default Modifier(không chỉ định rõ loại nào), thì trong cùng một package các đối tượng có thể truy suất tới các thuộc tính này:

```
fun main(args: Array<String>) {
    var teo=SinhVien()
    teo.ma=114
    teo.ten="Nguyễn Thị Tèo"
    println("Thông tin của Tèo:")
    println("Mã =" +teo.ma)
    println("Tên=" +teo.ten)
}
```

Để đảm bảo tính đóng gói thì các thuộc tính nên khai báo private, khi thuộc tính khai báo là private. Lúc này các đối tượng không thể truy suất trực tiếp vào các thuộc tính được mà phải thông qua getter/setter:

```
class SinhVien {
    private var ma:Int=0
```

```
    private var ten:String=""  
}
```

Khi khai báo Modifier là private thì ta không thể lấy tên đối tượng truy suất các thuộc tính được, tức là khai báo như dưới đây **sẽ bị báo lỗi**:

```
fun main(args: Array<String>) {  
    var teo=SinhVien()  
    teo.ma=114  
    teo.ten="Nguyễn Thị Tèo"  
}
```

5) Khai báo các getter/setter của Lớp trong Kotlin:

Nếu các thuộc tính trong lớp mà để default và truy suất trong cùng package hoặc các thuộc tính để modifier là public thì ta không cần khai báo getter/setter.

Cách khai báo getter/setter trong Kotlin có khác biệt so với Java, C# và các ngôn ngữ khác.

Cú pháp tổng quát để khai báo getter/setter:

```
var <propertyName>[: <.PropertyType>] [= <property_initializer>]  
[<getter>]  
[<setter>]
```

Ví dụ:

```
class SinhVien {  
    private var ma:Int=0  
    private var ten:String=""  
    public var Ma:Int  
        get()  
        {  
            return ma  
        }  
        set(value)  
        {  
            ma=value  
        }  
    public var Ten:String  
        get()  
        {  
            return ten  
        }
```

```

        }
        set(value)
        {
            ten=value
        }
    constructor()
    {
        println("Đây là secondary constructor 0 đối số")
    }
    constructor(ma:Int,ten:String)
    {
        println("Đây là secondary constructor 2 đối số")
        println("Mã=$ma ; Tên =$ten")
    }
}

```

Theo cách viết ở trên ta có 2 Getter/Setter Ma và Ten. Các đối tượng ở ngoài sẽ không truy suất được vào 2 thuộc tính **ma** và **ten** mà chỉ được truy suất thông qua 2 Getter/Setter **Ma** và **Ten** (chú ý IN HOA- in thường)

Các từ khóa get, set ở trên thì phần mềm IntelliJ IDEA cũng gợi ý sẵn cho ta, chỉ cần gõ ký tự đầu là nó gợi ý rồi sau đó ta nhấn Enter tự nó xuất hiện. Với set bạn để ý có value là parameter mặc định, đó là parameter nhận giá trị input từ ngoài vào (là thay đổi thông tin cho thuộc tính của đối tượng).

Ví dụ triệu gọi lớp SinhVien:

```

fun main(args: Array<String>) {
    var hanh=SinhVien()
    hanh.Ma=113
    hanh.Ten="Hồ Thị Hạnh"
    println("Thông tin của Hạnh:")
    println("Mã:"+hanh.Ma)
    println("Tên:"+hanh.Ten)
}

```

Kết quả khi chạy các đoạn lệnh ở trên:

```

Đây là secondary constructor 0 đối số
Thông tin của Hạnh:
Mã:113
Tên:Hồ Thị Hạnh

```

Có rất nhiều trường hợp để hiệu chỉnh cách viết get và set ở trên, khi nào gặp trường hợp cụ thể thì các bạn tìm hiểu thêm, Tui không có trình bày ra hết ở đây được.

6) Khai báo các Phương thức của Lớp trong Kotlin:

Phương thức hay còn gọi là hành vi, các xử lý nghiệp vụ trên đối tượng. Đây là tập hợp các sức mạnh của đối tượng. Kotlin dùng từ khóa **fun** để khai báo các phương thức. Các phương thức này có thể có giá trị trả về hoặc không.

Ví dụ dưới đây bổ sung 2 phương thức (trả về giá trị String và không trả về giá trị nào cả):

```
class SinhVien {
    private var ma:Int=0
    private var ten:String=""
    public var Ma:Int
        get()
        {
            return ma
        }
        set(value)
        {
            ma=value
        }
    public var Ten:String
        get()
        {
            return ten
        }
        set(value)
        {
            ten=value
        }
    constructor()
    {
        println("Đây là secondary constructor 0 đối số")
    }
    constructor(ma:Int,ten:String)
    {
        println("Đây là secondary constructor 2 đối số")
        println("Mã=$ma ; Tên =$ten")
    }
    fun xuatThongTin()
    {
        println("Thông tin chi tiết:")
    }
}
```

```

        println("Mã = "+ma)
        println("Tên= "+ten)
    }
    fun chiTiet():String
    {
        var s="Thông Tin Chi Tiết:"
        s=s.plus("\nMã="+ma)
        s=s.plus("\n")
        s=s.plus("Tên="+ten)
        return s
    }
}

```

xuatThongTin() là phương thức không trả về giá trị (còn gọi là các thủ tục) nó thực hiện nội tại bên trong Lớp. chiTiet() là phương thức trả về giá trị, ở ngoài khi truy suất có thể lưu lại kết quả của hàm này để sử dụng cho các mục đích khác (rất thường xuyên sử dụng trường hợp này).

Ví dụ dưới đây minh họa cách thức sử dụng 2 hàm trên:

```

fun main(args: Array<String>) {
    var hanh=SinhVien()
    hanh.Ma=113
    hanh.Ten="Hồ Thị Hạnh"
    println("Thông tin của Hạnh:")
    println("Mã:"+hanh.Ma)
    println("Tên:"+hanh.Ten)

    println("-----Gọi phương thức xuatThongTin() -----")
    hanh.xuatThongTin()
    println("-----Gọi phương thức chiTiet() -----")
    var detail=hanh.chiTiet()
    println(detail)
}

```

Kết quả khi chạy ta thấy:

Đây là secondary constructor 0 đối số
Thông tin của Hạnh:
Mã:113
Tên:Hồ Thị Hạnh
---Gọi phương thức xuatThongTin() ---
Thông tin chi tiết:
Mã = 113
Tên= Hồ Thị Hạnh

—Gọi phương thức chiTiet()—

Thông Tin Chi Tiết:

Mã=113

Tên=Hồ Thị Hạnh

Kotlin cũng giống như C# và Java, nó hỗ trợ một phương thức đặc biệt đó là `toString()` dùng để tự động xuất thông tin khi đối tượng được xuất ra màn hình/giao diện. Để tạo phương thức này rất đơn giản, trong lớp ta chỉ cần gõ từ khóa `to` thì phần mềm tự hiển thị hàm `toString` cho ta:

The screenshot shows the Android Studio code editor with two tabs: "SinhVien.kt" and "app_test_sinhvien.kt". The "SinhVien" tab is active, displaying the following code:

```
42     var s="Thông Tin Chi Tiết:"  
43         s=s.plus("\nMã="+ma)  
44         s=s.plus("\n")  
45         s=s.plus("Tên="+ten)  
46     return s  
47 }  
48 to  
49 } override fun toString(): String {...}
```

A red arrow points from the word "to" in line 48 to the "toString" part of the completion suggestion "override fun toString(): String {...}" in line 49. A tooltip below the suggestion bar says "Ctrl+Down and Ctrl+Up will move caret down and up in the editor".

Nó sẽ tự động ra hàm sau:

```
override fun toString(): String {  
    return super.toString()  
}
```

Ta muốn xuất thông tin gì thì cứ return đúng thông tin đó trong này, ví dụ:

```
override fun toString(): String {  
    var s="Thông Tin Chi Tiết:"  
    s=s.plus("\nMã="+ma)  
    s=s.plus("\n")  
    s=s.plus("Tên="+ten)  
    return s  
}
```

Trong main, chỉ cần xuất đối tượng là nó tự động triệu gọi hàm `toString()`, rất lợi hại. Đối tượng chứa rất nhiều thông tin và các mối quan hệ, còn `toString()` chỉ là giúp ta xuất

một vài thông tin cần thiết, nó giúp ta ẩn dấu được nhiều thông tin khác mà vẫn đảm bảo được tính chất của đối tượng:

```
fun main(args: Array<String>) {
    var hanh = SinhVien()
    hanh.Ma = 113
    hanh.Ten = "Hồ Thị Hạnh"
    println("Xuất thông tin đối tượng->tự gọi toString()")

    println(hanh)
}
```

Kết quả khi chạy ta thấy:

```
Đây là secondary constructor 0 đối số
Xuất thông tin đối tượng->tự gọi toString()
Thông Tin Chi Tiết:
Mã=113
Tên=Hồ Thị Hạnh
```

Như vậy tới đây Tui đã trình bày xong cách thức tạo cấu trúc của 1 Lớp trong Kotlin, bao gồm các quy tắc tạo:

- Tên Lớp
- Các constructors
- Các thuộc tính
- Các getter/setter
- Các phương thức

Các bạn chú ý học kỹ, thực hành lại và có gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ tiếp tục trình bày về OOP trong Kotlin (các loại phương thức, overloading method, tham chiếu this), các bạn chú ý theo dõi

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/eqs3538mpwoplml2/HocOOPPhan2.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 24-Lập trình hướng đối tượng trong Kotlin – phần 3

Trong [bài 23](#) Tui đã trình bày các quy tắc tạo lớp, constructor, thuộc tính, getter/setter, phương thức. Trong bài này Tui tiếp tục trình bày về các Loại phương thức, tham chiếu this và Kỹ thuật Overloading Method trong Kotlin.

1) Các loại phương thức trong OOP:

Trong lập trình hướng đối tượng, Người ta chia phương thức ra làm 2 loại **Service method** và **Support method**.

Service Method là các phương thức public để cung cấp ra ngoài cho các đối tượng sử dụng

Support Method là các phương thức private dùng để hỗ trợ cho các Service method. Các đối tượng ở ngoài không thể truy suất tới các Support Method

Ví dụ:

```
class TamGiac {  
    private var canhA: Double = 0.0  
    private var canhB: Double = 0.0  
    private var canhC: Double = 0.0  
  
    public var CanhA:Double  
        get() {return canhA}  
        set(value) {canhA=value}  
    public var CanhB:Double  
        get() {return canhB}  
        set(value) {canhB=value}  
    public var CanhC:Double  
        get() {return canhC}  
        set(value) {canhC=value}  
  
    /**  
     * Đây là service method  
     */  
    public fun chuVi(): Double {  
        return canhA + canhB + canhC.toDouble()  
    }  
}
```

```

    /**
     * Đây là support method
     */
    private fun nuaChuVi(): Double {
        return chuVi() / 2
    }
    /**
     * Đây là service method
     */
    public fun dienTich(): Double {
        val p = nuaChuVi()
        return Math.sqrt(p * (p - canhA) * (p - canhB) * (p - canhC))
    }
}

```

Ở Trên bạn quan sát phương thức nuaChuVi() là support method vì nó đ^ế private, phương thức này s^ẽ hỗ trợ các Service method khác như: dienTich()

Ở Ngoài chỉ có th^ể truy suất được các phương thức là Service Method, kh^{ông} th^ể truy suất được tới Support Method, ví dụ:

```

fun main(args: Array<String>) {
    var tg1= TamGiac ()
    tg1.CanhA=12.0
    tg1.CanhB=17.0
    tg1.CanhC=14.0
    var dt=tg1.dienTich()
    var cv=tg1.chuVi()

//var nucv=tg1.nuaChuVi() //lệnh này sẽ báo lỗi vì nuaChuVi() ko
thể truy suất
    println("Diện tích = $dt")
    println("Chu vi = $cv")
}

```

Kết quả khi chạy ta thấy:

```

Diện tích =83.0267276243018
Chu vi = 43.0

```

2)Tham chiếu this:

Tham chiếu **this** rất quan trọng trong các ngôn ngữ lập trình, C#, Java...cũng sử dụng tham chiếu **this**. Để hiểu được tham chiếu **this** ta cần đi từ khái niệm **Instance variable** và **local variable**

Instance variable là các thuộc tính, các biến khai báo ngoài lớp. Toàn bộ các hàm trong lớp điều có thể truy suất được.

Local variable là các biến được khai báo trong đối số của hàm hoặc nội dung hàm. Chỉ có hàm này mới truy suất được các biến local variable của nó, các đối số trong hàm thường được mặc định là `readOnly(val)`

Hình dưới đây sẽ minh họa 2 loại biến trên:

The diagram shows a code snippet for a class named `HinhTron`. It highlights two types of variables: **Instance variable** and **Local variable**.

```
4 class HinhTron {
5     private var banKinh: Double = 0.0
6     public fun doiBanKinh(banKinh: Double)
7     {
8         var bk = banKinh
9         if(bk<0.0)
10            bk=0.0
11         this.banKinh = banKinh
12     }
13     public fun dienTich(): Double
14     {
15         return Math.PI*Math.pow(this.banKinh, 2.0)
16     }
17     public fun chuVi(): Double
18     {
19         return 2*Math.PI*this.banKinh
20     }
21 }
```

- Instance variable:** The variable `banKinh` declared at line 5 is highlighted with a red box and labeled "Instance variable". It is used in the constructor and in the implementation of the `dienTich()` and `chuVi()` methods.
- Local variable:** The variable `bk` declared at line 8 is highlighted with a blue box and labeled "Local variable". It is used within the `doiBanKinh()` method to store a temporary value of `banKinh` before setting it to zero if it's negative.

Rõ ràng ta thấy biến `banKinh` ở dòng lệnh số 5 là instance variable. Biến `banKinh` ở dòng số 6 và biến `bk` ở dòng 8 là local variables.

Khi chúng ta rèn các quy tắc về OOP, ta có thể dùng công cụ để tạo ra các constructor, các hàm. Rất thường xuyên chúng ta gặp trường hợp instance variable và local variable trùng tên nhau. Vậy làm sao để phân biệt được là chương trình đang gọi loại biến nào. Ví dụ ta quan sát dòng lệnh số 11 ở trên:

this.banKinh=banKinh

Chúng ta có lưu ý quan trọng sau:

Nếu tại một dòng lệnh mà đồng thời cùng truy suất tới instance variable và local variable(cùng tên) thì chương trình sẽ ưu tiên truy suất tới biến local variable

Tức là nếu như dòng lệnh số 11 ta xóa từ khóa this đi, sẽ trở thành:

```
4 class HinhTron {
5     private var banKinh:Double=0.0
6     public fun doiBanKinh(banKinh: Double)
7     {
8         var bk=banKinh
9         if(bk<0.0)
10            bk=0.0
11            banKinh = banKinh
12    }
13    public fun dienTich():Double
14    {
15        return Math.PI*Math.pow(this.banKinh,2.0)
16    }
17    public fun chuVi():Double
18    {
19        return 2*Math.PI*this.banKinh
20    }
21 }
22
```

Vì như đã nói ở trên, Nếu tại một vị trí mà cùng truy suất tới instance và local cùng tên thì chương trình sẽ ưu tiên xử lý cho local. Tức nó nó sẽ cùng tham chiếu tới biến local. Như hình trên Tui vẽ 2 biến banKinh ở dòng 11 cùng trỏ tới biến banKinh ở dòng 6=> Như vậy rõ ràng nó sinh ra lỗi (lỗi thứ nhất là tham chiếu sai ý định, lỗi thứ 2 là mặc định các biến khai báo trong đối số của hàm thì Kotlin cho là readOnly(val) nên không thể đổi giá trị được).

Vì vậy trong trường hợp này bắt buộc ta phải dùng từ khóa **this** nhằm chỉ thị rõ cho chương trình : đây là đối tượng hiện tại trong lớp, phải truy suất tới Instance Variable. Do đó ta phải dùng từ khóa **this** là vậy, lúc này tham chiếu sẽ như sau:

```
4 class HinhTron {
5     private var banKinh:Double=0.0
6     public fun doiBanKinh(banKinh: Double)
7     {
8         var bk=banKinh
9         if(bk<0.0)
10            bk=0.0
11            this.banKinh = banKinh
12    }
13    public fun dienTich():Double
14    {
15        return Math.PI*Math.pow(this.banKinh,2.0)
16    }
17    public fun chuVi():Double
18    {
19        return 2*Math.PI*this.banKinh
20    }
21 }
```

3) Kỹ thuật Overloading Method

Overloading Method là một trong những kỹ thuật rất quan trọng trong lập trình OOP, Kotlin cũng hỗ trợ tính năng này.

- Overloading Là đặc điểm trong cùng 1 lớp có nhiều phương thức cùng tên nhưng khác nhau về Signature.
- Signature bao gồm: Số lượng các đối số hoặc kiểu dữ liệu các đối số hoặc thứ tự các đối số.
- Kiểu dữ liệu trả về không được tính vào signature
- Lợi ích của Overloading là khả năng tái sử dụng lại phương thức và giúp việc gọi hàm “uyển chuyển”.
- Các Constructor là trường hợp đặc biệt của Overloading Method

Ví dụ:

```
4 class SanPham {  
5     private var ma:Int=0  
6     private var ten:String?=null  
7     private var gia:Double=0.0  
8     constructor() → 0 đối số  
9     constructor(ma: Int, ten: String?, gia: Double) {  
10        this.ma = ma  
11        this.ten = ten  
12        this.gia = gia  
13    }  
14    constructor(ma: Int, ten: String?) {  
15        this.ma = ma  
16        this.ten = ten  
17    }  
18    fun printInfor() → 0 đối số  
19    {  
20    }  
21    fun printInfor(ma: Int, ten: String?, gia: Double)  
22    {  
23    }  
24 }
```

Đánh dấu bằng màu đỏ: 0 đối số, 3 đối số, 2 đối số, 3 đối số

ở ví dụ trên bạn thấy constructor Tui tạo có 3 constructor với số lượng các đối số khác nhau. Và có 2 hàm printInfor với số lượng đối số khác nhau. Đây chính là 2 ví dụ là Overloading Method ==> Giống Tên nhưng khác Signature

Khi chúng ta truy suất tới các phương thức này, thì dựa vào thông số đầu vào mà chương trình tự động điều hướng gọi chính xác phương thức.

```
fun main(args: Array<String>) {
    //Constructor 0 đối số được gọi
    var coca=SanPham()
    //Constructor 2 đối số được gọi
    var pepsi=SanPham(1,"Pepsi")
    //Constructor 3 đối số được gọi
    var biaSaiGon=SanPham(2,"Bia Sài Gòn",15.5)
    //Phương thức 0 đối số được gọi
    pepsi.printInfor()
    //Constructor 3 đối số được gọi
    coca.printInfor(5,"Cô Ca Cô La la",25.0)
}
```

Chắc chắn trong quá trình Coding chúng ta sẽ sử dụng overloading rất là nhiều vì lợi ích to lớn của nó, bài học này giúp các bạn biết được cách cài đặt kỹ thuật overloading, khi vào dự án cụ thể bạn có thể áp dụng.

Ngoài ra Kotlin còn hỗ trợ một loại Overloading Method đặc biệt đó là **vararg** (còn gọi là **Parameter list**) . Đây chính là trường hợp đặc biệt của Overloading Method (signature là số lượng các đối số khác nhau). Khi khai báo **vararg** ta có thể truyền bao nhiêu đối số vào hàm cũng được.

Ví dụ:

```
fun printNumbers(vararg numbers: Int) {
    for (number in numbers) {
        println(number)
    }
}

fun main(args: Array<String>) {
    printNumbers(1)
    printNumbers(7,8)
    printNumbers(9,10,11,12)
}
```

Ở trên hàm **printNumbers** là 1 Overloading Method đặc biệt, ta có thể truyền bao nhiêu đối số cũng được.

Kết quả khi chạy ta thấy:

1
7
8
9
10
11
12

Như vậy tới đây Tui đã trình bày xong các loại phương thức trong Kotlin, tham chiếu this, kỹ thuật Overloading Method. Đây là một trong kiến thức rất quan trọng trong lập trình OOP và Kotlin. Các bạn chú ý học kỹ, thực hành lại và có gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ tiếp tục trình bày về OOP trong Kotlin (Data Classes, Nested Class, Enum Class trong Kotlin), các bạn chú ý theo dõi

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/9t1b3sji51dscl5/HocOOPPhan3.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 25-Lập trình hướng đối tượng trong Kotlin – phần 4

Trong [bài 24](#) Tui đã trình bày chi tiết các loại phương thức, tham chiếu this, overloading, parameter list... là một trong những kỹ thuật rất quan trọng trong lập trình hướng đối tượng. Trong bài này Tui tiếp tục trình bày về [Data Classes, Nested Classes, Inner Classes, Enum Classes trong Kotlin](#) là một trong những kỹ thuật khá thú vị trong lập trình hướng đối tượng và Kotlin.

1) Data Classes

Trong quá trình xử lý, ta rất thường xuyên chỉ cần lưu trữ dữ liệu mà không có các phương thức. Kotlin hỗ trợ chức năng này bằng cách giúp ta tạo một lớp đặc biệt, lớp này gọi là Data Class.

Các Data Class trong Kotlin sẽ tự động cung cấp:

- equals() / hashCode()
- toString()
- componentN()
- copy()

Ví dụ:

```
fun main(args: Array<String>) {  
    data class User(var UserName:String, var Password:String)  
    var user1=User(UserName = "obama", Password = "113@114Xa")  
    var user1Copy=user1.copy()  
    var user2Copy=user1.copy(Password ="cohangxom" )  
    println(user1.toString())  
    println(user1Copy.toString())  
    println(user2Copy.toString())  
    println("User Name của user 1="+user1.UserName)  
}
```

Ở dòng số 5, ta thấy một Data Class tên là [User](#) được tạo ra, nó có 2 thuộc tính là [UserName](#) và [Password](#)

Kết quả khi chạy ta thấy:

```
User (UserName=obama, Password=113@114Xa)
User (UserName=obama, Password=113@114Xa)
User (UserName=obama, Password=cohangxom)
User Name của user 1=obama
```

2) Nested Class

Kotlin giống như các ngôn ngữ lập trình khác, đó là khả năng cho phép Lớp này nằm trong lớp khác dạng Nested. Cần lưu ý là các lớp Nested sẽ không thể truy suất được các biến thành viên trong Outer class

Ví dụ:

```
class Outer {
    private val bar: Int = 1
    class Nested {
        fun foo() = 113
    }
}
```

Ở ví dụ trên thì phương thức foo() trong Nested class không thể truy suất được tới biến **bar** trong Outer class

Khi triệu khai foo() ta làm như sau:

```
fun main(args: Array<String>) {
    val demo = Outer.Nested().foo()
    println(demo)
}
```

Chạy chương trình ta sẽ được kết quả là **113** xuất ra màn hình.

3) Inner Classes

Kotlin giống như các ngôn ngữ lập trình khác, đó là khả năng cho phép Lớp này nằm trong lớp khác dạng Inner. Cần lưu ý là các lớp Inner sẽ có thể truy suất được các biến thành viên trong Outer class

```
package inner

class Outer {
    private val bar: Int = 1
```

```

inner class Inner {
    fun foo() = bar
}
}

```

Code ở trên ta thấy dùng từ khóa **inner** đằng trước class Inner

Hàm main sử dụng như sau:

```

package inner

import inner.Outer

fun main(args: Array<String>) {
    val demo = Outer().Inner().foo()
    println(demo)
}

```

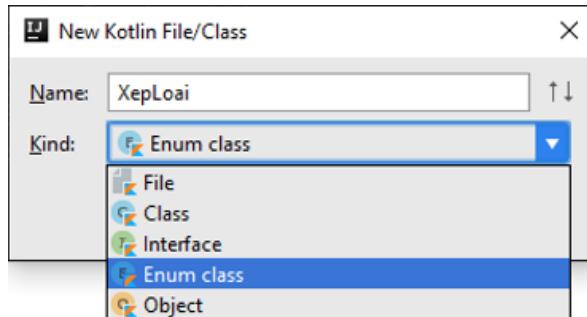
Chạy chương trình ta sẽ được kết quả là 1 xuất ra màn hình.

Ở đây ta để ý có sự khác biệt giữa Nested Class và Inner class cả cách khai báo cũng như cách truy suất (**hãy tự để ý kỹ nó khác chỗ nào**).

4)Enum Classes

Enum cũng là một loại Lớp đặc biệt trong Kotlin (giống như các ngôn ngữ khác C#, java...)

Cách khai báo Enum trong Kotlin rất đơn giản, tương tự như các ngôn ngữ lập trình khác.



Lúc new File ta chọn Kind là Enum class, Name là nơi đặt tên Enum. Ví dụ đặt là XepLoai:

```
enum class XepLoai {
    XuatSac, Gioi, Kha, TrungBinh, Yeu, Kem
}
```

Ta có thể sử dụng Enum này như sau:

```
class SinhVien {
    var Ma:Int=0
    var Ten:String=""
    var DiemTrungBinh:Double=0.0
    public fun XepLoaiHocTap():XepLoai
    {
        if(DiemTrungBinh>=9) return XepLoai.XuatSac
        if(DiemTrungBinh>=8) return XepLoai.Gioi
        if(DiemTrungBinh>=7) return XepLoai.Kha
        if(DiemTrungBinh>=5) return XepLoai.TrungBinh
        if(DiemTrungBinh>=3) return XepLoai.Yeu
        return XepLoai.Kem
    }
}
```

Tạo main để triệu hồi như sau:

```
fun main(args: Array<String>) {
    var teo=SinhVien()
    teo.DiemTrungBinh=9.0
    println(teo.XepLoaiHocTap())
}
```

Chạy chương trình ta sẽ có kết quả là **XuatSac**.

Như vậy tới đây Tui đã trình bày xong các loại class đặc biệt trong Kotlin(Data Classes, Nested Classes, Inner Classes và Enum classes). Các bạn chú ý học kỹ, thực hành lại và có gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ tiếp tục trình bày về OOP trong Kotlin (Kết thừa rất quan trọng), các bạn chú ý theo dõi.Các bạn có thể tải source code bài này ở đây: <http://www.mediafire.com/file/8r7z95j8r9lptob/HocOOPPhan4.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 26-Lập trình hướng đối tượng trong Kotlin – phần 5

Trong phần này Tui sẽ trình bày các kiến thức liên quan tới Kế Thừa trong lập trình Hướng đối tượng. Để học tốt bài này thì bắt buộc bạn phải đã thông kinh mạch các bài:

[Bài 22-Lập trình hướng đối tượng trong Kotlin – phần 1](#)

[Bài 23-Lập trình hướng đối tượng trong Kotlin – phần 2](#)

[Bài 24-Lập trình hướng đối tượng trong Kotlin – phần 3](#)

[Bài 25-Lập trình hướng đối tượng trong Kotlin – phần 4](#)

Trước khi đi vào kỹ thuật cài đặt Kế Thừa trong Kotlin, Tui điểm sơ sơ 1 xíu về Khái Niệm Kế Thừa.

Kế thừa là gì?

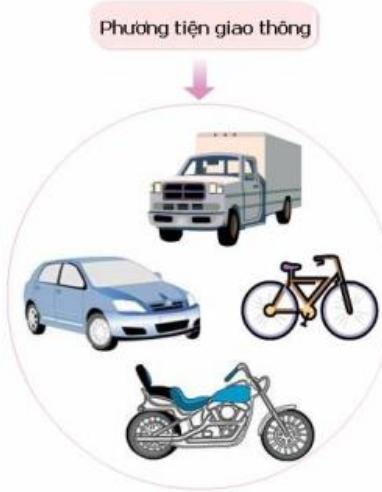
Kế thừa bạn có thể hiểu nôm na là **SỰ TÁI SỬ DỤNG** lại mã nguồn cũ, có thể mở rộng thêm mã lệnh mới từ những thứ đã tồn tại, giúp nâng cao hiệu suất lập trình.

Thường các đối tượng có cùng chung một số đặc điểm, hành vi được nhóm lại với nhau.

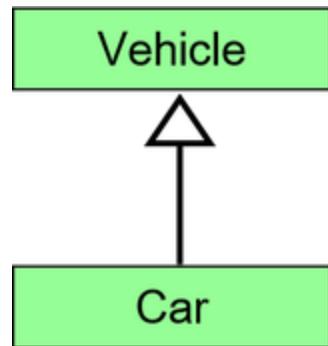
Ví dụ:

- Xe đẹp
- Xe máy
- Xe hơi
- Xe tải

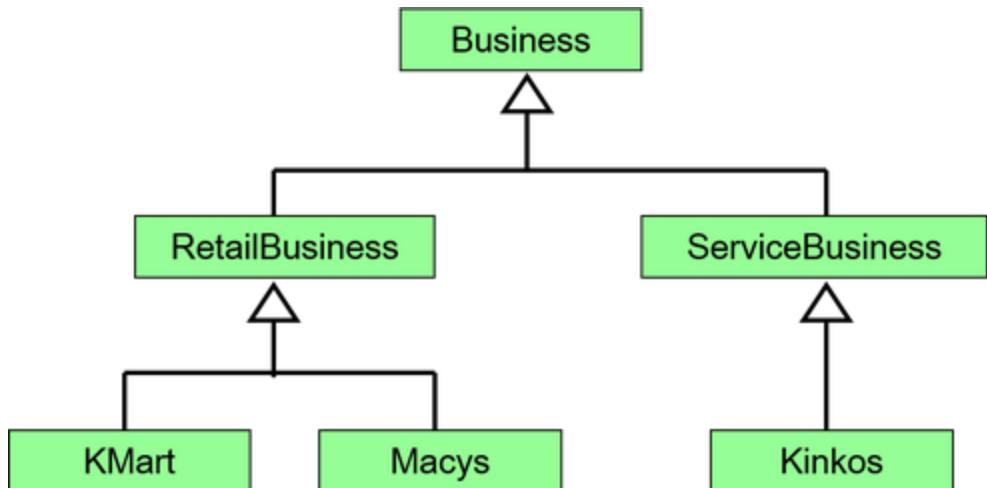
các xe này có một số đặc điểm giống nhau đúng ko?



Ta thường thấy mô hình Lớp kế thừa người ta vẽ giống như dưới đây:



Hay Một lớp con có thể là lớp cha của các lớp khác:



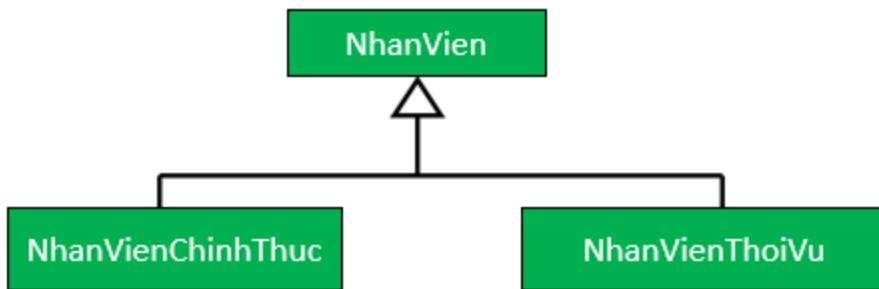
Để làm tốt được Kế Thừa thì ta cần biết 2 khái niệm:

Tổng quát hóa: Những đặc điểm chung mà các lớp đều có=>là các lớp Cha

Chuyên biệt hóa: Những đặc điểm riêng chỉ có các lớp con mới có =>là các lớp Con

Ví dụ: Công ty có 2 loại nhân viên (Nhân Viên Chính Thức và Nhân Viên Thời Vụ):

Thì rõ ràng Nhân viên không kể là Chính thức hay thời vụ thì mọi người đều có mã, tên. Nhưng Nhân viên chính thức và nhân viên thời vụ thì khác nhau ít nhất là cách tính lương. Do đó Lớp Tổng quát Hoá là lớp Nhân Viên vì nó có các đặc điểm chung của Chính Thức và Thời Vụ, Còn các lớp chuyên biệt hóa là: Nhân Viên Chính Thức, Nhân Viên Thời Vụ:



Cài đặt Kế thừa trong Kotlin như thế nào?

Ta sẽ viết ví dụ bài Nhân Viên theo hình ở trên. Trong Kotlin, để viết Lớp cho phép kế thừa ta làm như sau:

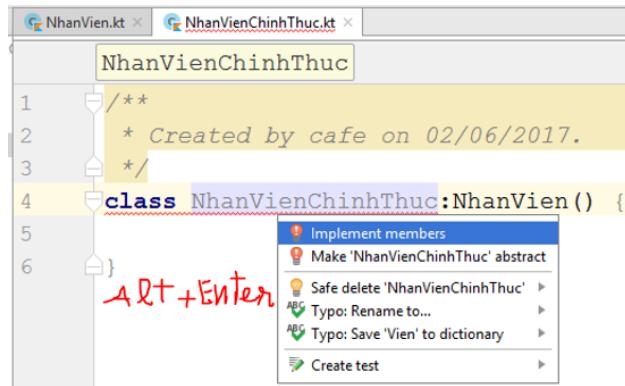
Lớp Tổng quát hóa (lớp cha) Nhân Viên:

```
open abstract class NhanVien {  
    protected var ma:Int=0  
    protected var ten:String=""  
    public abstract fun tinhLuong(ngayCong:Int):Double  
}
```

Lớp cha NhanVien ở trên là một lớp trừu tượng, khi có phương thức trừu tượng tinhLuong. Vấn đề là làm sao biết được nên khai báo nó là phương thức trừu tượng? Ta hiểu nôm na như sau: Mọi nhân viên đều tính lương, nhưng ngay thời điểm này không biết tính lương cho nhân viên nào, mà không biết tính cụ thể cho ai thì khai báo nó là trừu tượng (tuy không biết là tinhLuong cho đối tượng nào nhưng chắc chắn nó phải có tinhLuong). Với abstract hay interface thì các hàm trừu tượng nó giống như là các luật, các mẫu đưa ra và yêu cầu các lớp con kế thừa phải tuân thủ theo (override lại toàn bộ).

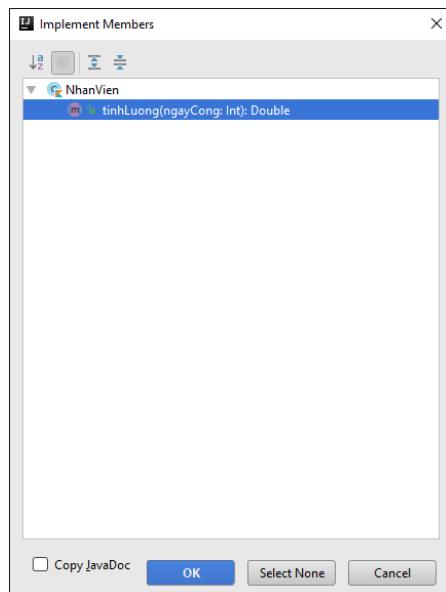
Một điều cần lưu ý nữa là Nếu muốn các lớp khác có thể kế thừa từ lớp này thì bắt buộc ta phải thêm từ khóa **open** đằng trước khai báo lớp như mẫu code ở trên.

Bây giờ ta tạo 1 lớp NhanVienChinhThuc kế thừa từ lớp NhanVien như sau:



Bạn thấy ta dùng dấu 2 chấm (:) để kế thừa nhé, nhìn vào hình trên là ta biết cách viết NhanVienChinhThuc kế thừa từ NhanVien phải viết như thế nào.

Ở trên bạn thấy nó báo lỗi màu đỏ, thực ra là do lớp NhanVien là lớp trừu tượng. Thì các lớp con kế thừa từ lớp trừu tượng bắt buộc phải Override lại toàn bộ phương thức trừu tượng của lớp cha. IntelliJ IDEA hỗ trợ chúng ta công cụ overrided này rất nhanh chóng. Đó là ngày chỗ báo lỗi bạn nhấn tổ hợp phím Alt+Enter nó sẽ xuất hiện ra màn hình có dòng **Implement members** như hình trên, ta chọn nó rồi nhấn Enter, màn hình Implement Members sẽ xuất hiện như dưới đây:



Ta chọn các phương thức trừu tượng của lớp cha rồi bấm OK, nó sẽ tự động phát sinh Coding như sau:

```
class NhanVienChinhThuc:NhanVien() {  
    override fun tinhLuong(ngayCong: Int): Double {  
        TODO("not implemented") //To change body of created  
        functions use File | Settings | File Templates.  
    }  
}
```

Giả sử rằng nhân viên chính thức có lương: Nếu $\text{NgayCong} \geq 22$ thì lương là 10 triệu, còn < 22 ngày thì lương 10 triệu – mỗi ngày nghỉ mất 500 nghìn, ta sửa lại coding như sau:

```
class NhanVienChinhThuc:NhanVien() {  
    override fun tinhLuong(ngayCong: Int): Double {  
        if(ngayCong>=22)  
            return 10000000.0  
        return 10000000.0-500000*(22-ngayCong)  
    }  
}
```

Tương tự như vậy ta Tạo lớp NhanVienThoiVu kế thừa từ NhanVien. Và lương của Nhân viên thời vụ là $150k * \text{số ngày công}$:

```
class NhanVienThoiVu:NhanVien() {  
    override fun tinhLuong(ngayCong: Int): Double {  
        return ngayCong*150000.0  
    }  
}
```

Cách sử dụng các lớp Kế thừa trong Kotlin như thế nào?

Tạo một tập tin Kotlin để thử nghiệm

```
fun main(args: Array<String>) {  
    var an=NhanVienChinhThuc()  
    var binh=NhanVienThoiVu()  
    var luongAn=an.tinhLuong(20)  
    println("Lương của An=" +luongAn)  
    var luongBinh=binh.tinhLuong(3)
```

```
    println("Lương của Bình=" + luongBinh)  
}
```

Kết quả khi chạy ta thấy:

```
Lương của An=9000000.0  
Lương của Bình=450000.0
```

Ở trên các bạn thấy Tui hay nhắc tới Override, vậy nó là cái gì? (bạn đã từng học Overloading đúng ko), bạn có thể hiểu nôm na Overriding Method như sau:

- Trong một tập các lớp có mối quan hệ huyết thống có các phương thức giống signature y xì (nội dung phương thức khác nhau)
- Overriding methods giúp lập trình viên có thể định nghĩa cách hành xử khác nhau ứng với các đối tượng khác nhau nhưng cùng sử dụng một tên phương thức.
- Ví dụ: Nhân viên chính thức và Nhân viên thời vụ đều có phương thức là Tính Lương, tuy nhiên cách thức tính lương của 2 đối tượng này sẽ khác nhau.

Bạn tự so sánh sự khác nhau giữa Overloading Method và Overriding Method.

Còn cách kế thừa nào khác trong Kotlin không?

Trong Kotlin còn có thể dùng Interface để Kế thừa (implements), Interface không phải là GUI mà là tập các giao ước (các luật, các Rule) khi các lớp con implements(hay gọi là kế thừa) thì phải định nghĩa lại toàn bộ các phương thức này.

Giả sử ta có interface:

```
interface MyInterface {  
    fun bar()  
    fun foo() {  
        // optional body  
    }  
}
```

Thì khi lớp con kế thừa ta có thể viết như sau:

```
class Child : MyInterface {  
    override fun bar() {  
        // body  
    }  
}
```

Chúng ta chú ý là với Kotlin kế thừa từ Lớp cũng chỉ cho đơn thừa kế giống như Java , C#. Kế thừa từ Interface cho phép đa thừa kế Interface (đa dãy xuất), vì vậy đôi khi nó bị conflict nếu như các Interface mà nó kế thừa có các phương thức trùng tên nhau. Trường hợp này Kotlin giải quyết như sau:

```
interface A {  
    fun foo() { print("A") }  
    fun bar()  
}  
interface B {  
    fun foo() { print("B") }  
    fun bar() { print("bar") }  
}  
class C : A {  
    override fun bar() { print("bar") }  
}  
class D : A, B {  
    override fun foo() {  
        super<A>.foo()  
        super<B>.foo()  
    }  
    override fun bar() {  
        super<B>.bar()  
    }  
}
```

Như vậy tới đây Tui đã trình bày xong phần kế thừa trong Kotlin, còn rất nhiều phần khác liên quan tới Kế thừa trong Kotlin, các bạn tự xử nhé (ví dụ như tính đa hình, seal classes). Các bạn chú ý học kỹ, thực hành lại và có gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ trình bày về alias và cơ chế gom rác tự động trong Kotlin, các bạn chú ý theo dõi

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/448pk2x9111n016/HocOOPPhan5.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 27-Alias và cơ chế gom rác tự động trong Kotlin-OOP phần 6

Như vậy hầu hết các bạn đã học xong hướng đối tượng cài đặt bằng Kotlin, trong bài này Tui sẽ nói thêm về Alias và cơ chế gom rác tự động trong Kotlin.

Alias là khả năng mà tại 1 ô nhớ có nhiều đối tượng cùng trả tới (≥ 2 đối tượng).

Cơ chế gom rác tự động còn gọi là **garbage collection**, nó tự động thu hồi bộ nhớ khi ô nhớ đó không còn đối tượng nào quan lý.

Hai khái niệm này vô cùng quan trọng, các bạn cần phải đảm bảo hiểu rõ nguồn gốc về nó để có thể kiểm soát vấn đề quản lý bộ nhớ khi thực thi các dự án.

Bây giờ Tui sẽ trình bày chi tiết quá trình tạo Alias và garbage collection trong Kotlin (các ngôn ngữ khác tương tự)

Vì Kotlin chạy trong nền JVM nên cơ chế hoạt động sẽ giống với Java.

Để cho có cảm giác về Alias và cơ chế gom rác tự động, ta tạo một lớp phân số như sau:

```
class PhanSo {
    var tu:Int=1
    var mau:Int=1

    constructor(tu: Int, mau: Int) {
        this.tu = tu
        this.mau = mau
    }
}
```

Giả sử chúng ta có 2 đối tượng psA, psB như dưới đây:

psA=PhanSo(1,5)

psB=PhanSo(3,7)

Lúc này trên thanh RAM sẽ có 2 ô nhớ cấp phát cho 2 đối tượng phân số được quản lý bởi 2 biến đối tượng psA và psB (Chú ý là phải tưởng tượng nha các thím, chứ mình làm sao biết ô nhớ nào trên thanh RAM được cấp phát):



Hình trên cho thấy 2 đối tượng psA và psB quản lý 2 ô nhớ độc lập. Tức là psA thao tác trên vùng nhớ A sẽ không ảnh hưởng gì tới psB và ngược lại.

Bây giờ Giả sử ta thực hiện lệnh:

psA=psB

Thì lệnh trên: Ngôn ngữ nói “Phân số A bằng Phân số B”, nhưng hệ thống máy tính sẽ làm việc theo cơ chế “Phân số A trả tới vùng nhớ mà phân số B đang quản lý”. Hay nói cách khác “Vùng nhớ B” bây giờ có 2 biến đối tượng cùng trả tới(cùng quản lý):



Như vậy đã xuất hiện Alias ở “vùng nhớ B”. Lúc này sẽ xảy ra 2 hiện tượng như sau:

- Tại “vùng nhớ B”, nếu psA thay đổi thông tin sẽ làm cho psB thay đổi thông tin (vì cả 2 đối tượng này cùng quản lý một vùng nhớ)
- “Vùng nhớ A” không còn đối tượng nào tham chiếu tới, lúc này hệ thống sẽ tự động thu hồi bộ nhớ (hủy vùng nhớ A đã cấp trước đó), cơ chế này gọi là cơ chế gom rác tự động

Hình sau minh họa GC (**garbage collection**):



Tức là trong hàm main ta có như sau:

```
fun main(args: Array<String>) {
    var psA=PhanSo(1,5)
    var psB=PhanSo(3,7)
    psA=psB
    println("Tử số của A="+psA.tu)
}
```

Nếu bạn đã hiểu những gì Tui giải thích ở trên thì: Dòng lệnh số 7 phát sinh ra 2 hiện tượng. Đầu tiên là Alias (psA và psB cùng trỏ tới 1 ô nhớ), Thứ hai là Cơ chế gom rác tự động sẽ tự động thu hồi bộ nhớ cấp cho psA ở dòng 5 (vì lúc này psA đã trỏ qua vùng nhớ của psB). Khi chạy hàm main ở trên thì kết quả là gì?

Tử số của A=3

Vấn đề là bạn có hiểu vì sao Tử số của A bằng 3 hay không? rõ ràng bên trên Khai báo psA=PhanSo(1,5) thì Tử số của psA là 1 chứ sao là 3 được? hi hi hi rõ ràng bạn phải hiểu dòng lệnh số 7 đã làm thay đổi điều đó rồi, lúc này psA đã trỏ qua vùng nhớ mà psB đang trỏ. Mà vùng nhớ psB đang trỏ thì tử số bằng mấy? nó bằng 3 chứ gì nữa ==> tử số của psA phải là 3 chứ không phải 1 vì psA đã trỏ qua vùng nhớ của psB rồi.... **do you understand?**

Tiếp tục nếu giả sử Tui sửa tiếp coding như sau:

```
fun main(args: Array<String>) {
    var psA=PhanSo(1,5)
    var psB=PhanSo(3,7)
    psA=psB
    println("Tử số của A="+psA.tu)
    psA.tu=113
    println("Tử số của B="+psB.tu)
}
```

Khi chạy hàm main ở trên thì kết quả là gì?

Tử số của A=3
Tử số của B=113

Bạn có hiểu vì sao Tử số của B=113 không? thật phi lý đúng không? rõ ràng dòng lệnh 9 mình chỉ đổi psA.tu=113, đâu có đổi tử số của phân số B đâu, tại sao tử số của B bị đổi theo A? Nó phải là 113 mới đúng, rất có lý không hề phi lý chút nào. Vì như trên Tui giải thích là vùng nhớ B có psA và psB cùng trả tới, do đó psA đổi cùng làm psB đổi và ngược lại, do đó khi psA.tu=113 tức là psB.tu cũng bằng 113. Các bạn cứ tưởng tượng thế này: Nếu Tui và các bạn cùng sử dụng chung 1 thẻ ATM, một ngày đẹp trời Tui buồn buồn Tui lấy hết sạch tiền trong ATM đó đi Nhậu -> bạn làm gì còn Cắc nào trong ATM đúng không?

Đôi khi trong quá trình thực hiện phần mềm ta có nhu cầu sao chép đối tượng ra (tạo thêm một đối tượng giống y xì đối tượng cũ nhưng nằm ở ô nhớ khác, để ta có thể tự do thay đổi thông tin trên đối tượng sao chép mà không làm ảnh hưởng tới đối tượng gốc). Kotlin hỗ trợ chúng ta hàm clone trong interface Cloneable để sao chép đối tượng:

Ta chỉnh là lớp Phân số như sau:

```
class PhanSo:Cloneable {
    var tu:Int=1
    var mau:Int=1
    constructor(tu: Int, mau: Int) {
        this.tu = tu
        this.mau = mau
    }
    fun copy():PhanSo
    {
        var ps:PhanSo=clone() as PhanSo
        return ps
    }
}
```

Ta thấy lớp Phân số kế thừa từ Cloneable và Tui có tự bổ sung hàm copy() cho lớp phân số. hàm này đơn thuần là gọi lệnh clone() của Cloneable để tạo ra 1 phiên bản mới của đối tượng (dữ liệu giống nhau y xì nhưng nằm trên ô nhớ khác nhau).

Trong main ta sửa lại như sau:

```

fun main(args: Array<String>) {
    var psA=PhanSo(1,5)
    var psB=PhanSo(3,7)
    psA=psB
    println("Tử số của A=" + psA.tu)
    psA.tu=113
    println("Tử số của B=" + psB.tu)
    var psC=psA.copy()
    psC.tu=114
    println("Tử số của A=" + psA.tu)
    println("Tử số của C=" + psC.tu)
    println("Mẫu số của C=" + psC.mau)
}

```

Bạn thấy dòng lệnh 8 Tui gọi hàm copy và lưu vào biến psC. Chú ý ngay chỗ này nó tạo ra thêm 1 ô nhớ mới và để cho psC quản lý. Tức là psC không liên can gì tới psA và psB (lúc này psA và psB vẫn đang cùng quản lý ô nhớ B). Như vậy khi chạy đoạn lệnh trên kết quả là gì?

Tử số của A=3
 Tử số của B=113
 Tử số của A=113
 Tử số của C=114
 Mẫu số của C=7

Bạn hiểu vì sao ra kết quả ở trên không? Tui chủ ý không giải thích nữa, để những bạn nào không hiểu thì Comment vào bài học này, những bạn hiểu sẽ giải thích cho các bạn. Nhưng Tui rất mong muốn bạn phải tự hiểu.

Như vậy tới đây Tui đã trình bày xong alias và Cơ chế gom rác tự động trong Kotlin. Các bạn chú ý học kỹ, thực hành lại và có gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ trình bày về Extension method trong Kotlin rất là hay và rất là mới mẻ. Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/xx1k62gbwk2n7pk/HocGarbageCollection.rar>

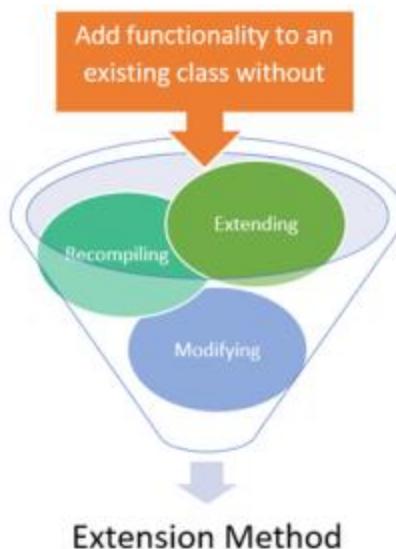
Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 28-Extensions Method trong Kotlin-OOP phần 7

Kotlin hỗ trợ Extensions Method rất tuyệt vời, giống như [LINQ trong C#\(ban nào quan tâm thì đăng ký học khóa học LinQ C# ở đây\)](#). Extensions Method cho phép ta chèn thêm phương thức vào các Lớp có sẵn mà không cần sửa mã nguồn, không cần biết mã nguồn, giống như là cài đặt virus vào một chương trình nào đó.



Cú pháp:

```
fun [Kiểu_Dữ_Liệu].Tên_Hàm([Các đối số]):[Kiểu_Trả_Về]  
{  
    this ở trong này là đối tượng thuộc [Kiểu_Dữ_Liệu]  
}
```

Khi khai báo như trên, thì bất kỳ [Kiểu_Dữ_Liệu] nào cũng có một hàm mới là Tên_Hàm

Ví dụ 1: Hãy cài đặt hàm Cong (Cộng) vào kiểu Int

```
fun Int.Cong (a:Int):Int  
{  
    return this+a
```

```

}
fun main(args: Array<String>) {
    var t=5.Cong(9)
    println("t=$t")
    var x1=9
    var x2=10
    var x3=x1.Cong(x2)
    println("x3=$x3")
}

```

Bạn nhìn vào dòng 4 Tui khai báo **Int.Cong** ==> là cài đặt hàm **Cong** vào kiểu **Int**
dòng lệnh số 6 có từ khóa **this**, **this** ở đây chính là đối tượng hiện tại của kiểu **Int**.

Bạn tiếp tục quan sát dòng số 9, thấy số 5.Cong(9) . Vì 5 là số Nguyên kiểu Int mà kiểu Int ta vừa cài đặt hàm Cong nên hiển nhiên 5 sẽ có phương thức Cong. Bạn để ý là Ta không hề biết Kotlin tao ra **Int** như thế nào nhưng ta vẫn có thể thêm phương thức Cong vào Int mà không cần biết mã nguồn cũ cũng không cần sửa mã nguồn cũ.

Khi chạy hàm main ở trên thì ta có kết quả sau:

```

t=14
x3=19

```

Ví dụ 2: Cài đặt hàm kiểm tra 1 số có phải là số Nguyên Tố hay không vào kiểu Int có sẵn của Kotlin

```

fun Int.Cong(a:Int):Int
{
    return this+a
}
fun Int.KiemTraNguyenTo():Boolean
{
    var dem=0
    for(i in 1..this)
    {
        if(this%i==0)
            dem++
    }
    return dem==2
}
fun main(args: Array<String>) {
    var t=5.Cong(9)
    println("t=$t")
}

```

```

var x1=9
var x2=10
var x3=x1.Cong(x2)
println("x3=$x3")

var a=7
if(a.KiemTraNguyenTo()==true)
{
    println("$a là số nguyên tố")
}
else
{
    println("$a Không là số nguyên tố")
}
var b=9
if(b.KiemTraNguyenTo()==true)
{
    println("$b là số nguyên tố")
}
else
{
    println("$b Không là số nguyên tố")
}
}

```

Ở trên cùng ta cài đặt hàm KiemTraNguyenTo() vào kiểu Int

Trong main hàm KiemTraNguyenTo() được tự động gắn vào dữ liệu kiểu Int

Khi chạy hàm main kết quả như sau:

```
t=14
x3=19
7 là số nguyên tố
9 Không là số nguyên tố
```

Khi chạy kết quả

Ví dụ 3:Cài đặt hàm Tính Tuổi cho Lớp Sinh Viên được viết sẵn của ai đó, Lớp Sinh Viên này có cấu trúc như sau:

```
import java.util.*

/*
 * Created by cafe on 02/06/2017.
 */
```

```

class SinhVien {
    private var ma:Int=0
    private var ten:String?=null
    private var namSinh:Date?=null
    public var Ma:Int
        get() {return ma}
        set(value) {ma=value}
    public var Ten:String?
        get()= ten
        set(value) {ten=value}
    public var NamSinh:Date?
        get() = namSinh
        set(value) {namSinh=value}
    constructor(ma: Int, ten: String?, namSinh: Date?) {
        this.ma = ma
        this.ten = ten
        this.namSinh = namSinh
    }
}

```

Vậy làm sao để cài đặt hàm tính tuổi cho lớp Sinh Viên này? (không được đổi mã nguồn cũ). Rất dễ dàng ta làm như sau:

```

import java.util.*

/**
 * Created by cafe on 02/06/2017.
 */
fun SinhVien.Tuoi():Int
{
    var cal=Calendar.getInstance()
    var yearHienTai=cal.get(Calendar.YEAR)
    cal.time=this.NamSinh
    var yearNamSinh=cal.get(Calendar.YEAR)
    return yearHienTai-yearNamSinh+1
}
fun main(args: Array<String>) {
    var ns=Calendar.getInstance()
    ns.set(Calendar.YEAR,1998)
    ns.set(Calendar.MONTH,2)
    ns.set(Calendar.DAY_OF_MONTH,15)
    var teo=SinhVien(1,"Nguyễn Văn Tèo",ns.time)
    var tuoiCuaTeo=teo.Tuoi()
    println("Tuổi của Tèo="+tuoiCuaTeo)
}

```

Ở trên ta cài đặt hàm Tuoi() vào Lớp Sinh Viên. Dựa vào 3 ví dụ ở trên mà bạn có thể áp dụng vào các dự án bất kỳ.

Khi chạy hàm main ở trên thì kết quả là gì?

Tuổi của Tèo=20

Như vậy tới đây Tui đã trình bày xong Extensions Method trong Kotlin. Các bạn chú ý học kỹ, thực hành lại và cố gắng hiểu được nó thông qua các ví dụ ở trên nhé. Các bài sau Tui sẽ trình bày về Xử lý File trong Kotlin rất quan trọng trong quá trình xử lý lưu trữ dữ liệu

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/3wd89egcx10qim7/HocExtension.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 29-Xử lý Text File trong Kotlin

Trong tất cả các ngôn ngữ lập trình thì xử lý file rất quan trọng, hầu như ta phải gặp trong các dự án. Tôi sẽ trình bày chuỗi 4 bài xử lý file: [Text File](#), [Serialize File](#), [XML File](#), [JSON File](#) để các bạn có nhiều lựa chọn trong quá trình xử lý tập tin.

Tại sao phải lưu trữ dữ liệu? Như chúng ta đã biết trong kiến trúc máy tính, một chương trình muốn hoạt động thì mọi tài nguyên phải được nạp lên bộ nhớ, cụ thể là trên thanh RAM. Mà nguyên lý của RAM là bộ nhớ tạm, khi tắt phần mềm, tắt máy... thì dữ liệu trong bộ nhớ sẽ không còn nữa. Giả sử chúng ta đang nhập liệu 100 Sản phẩm thì tự nhiên cúp điện, nếu không có cơ chế lưu dữ liệu từ bộ nhớ RAM xuống ổ cứng thì sẽ mất toàn bộ dữ liệu.

Text File là cách lưu trữ dữ liệu dạng thô, ta có thể mở tập tin lên xem cấu trúc, nội dung và chỉnh sửa được.

Kotlin dùng các thư viện JVM để tương tác File nên nó giống y xì như Java, từ Kotlin ta sẽ triệu hồi các thư viện Java. Do đó để hỗ trợ tốt cho Kotlin thì các bạn nên [đăng ký tham gia học Java](#) trước.

Để lưu và đọc Text File, tương tự như Java thì Kotlin sẽ phải import các thư viện sau:

```
import java.io.BufferedReader  
import java.io.OutputStreamWriter  
import java.io.FileOutputStream  
import java.io.BufferedWriter  
import java.io.InputStreamReader  
import java.io.FileInputStream
```

Giả sử ta có một cấu trúc dữ liệu (class) Sản phẩm như sau:

```
class SanPham {  
    var ma:Int=0  
    var ten:String=""  
    var donGia:Double=0.0  
    constructor()  
    constructor(ma: Int, ten: String, donGia: Double) {  
        this.ma = ma  
        this.ten = ten  
        this.donGia = donGia  
    }  
}
```

```

    override fun toString(): String {
        return "$ma\t$ten\t$donGia"
    }
}

```

Bây giờ ta sẽ viết Lớp để lưu danh sách Sản phẩm như sau (Lớp [TextFileFactory](#)):

```

import java.io.BufferedWriter
import java.io.OutputStreamWriter
import java.io.FileOutputStream
import java.io.BufferedReader
import java.io.InputStreamReader
import java.io.FileInputStream

/**
 * Created by cafe on 02/06/2017.
 */
class TextFileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ
     * @return true nếu lưu thành công, false nếu lưu thất bại
     */
    fun LuuFile(data:MutableList<SanPham>,path:String):Boolean
    {
        try {
            val fos = FileOutputStream(path)
            val osw = OutputStreamWriter(fos, "UTF-8")
            val bw = BufferedWriter(osw)
            for (sp in data) {
                bw.write(sp.toString());
                bw.newLine();
            }
            bw.close();
            osw.close();
            fos.close();
            return true
        }
        catch (ex:Exception)
        {
            ex.printStackTrace()
        }
        return false
    }
}

```

```

    * @author Trần Duy Thành
    * @param path:đường dẫn muốn đọc dữ liệu
    * @return Danh sách sản phẩm MutableList
    */
    fun DocFile(path:String) :MutableList<SanPham>
    {
        var data:MutableList<SanPham> = mutableListOf()
        try {
            val fis = FileInputStream(path)
            val isr = InputStreamReader(fis, "UTF-8")
            val br = BufferedReader(isr)

            var line = br.readLine()
            while (line != null) {
                var arr = line.split("\t")
                if (arr.size == 3) {
                    var sp: SanPham = SanPham()
                    sp.ma = arr[0].toInt()
                    sp.ten = arr[1]
                    sp.donGia = arr[2].toDouble()
                    data.add(sp)
                }
                line = br.readLine()
            }
            br.close()
            isr.close()
            fis.close()
        }
        catch (ex:Exception)
        {
            ex.printStackTrace()
        }
        return data
    }
}

```

Bây giờ ta tạo hàm main trong tập tin app_test_textfile.kt để test lưu và đọc danh sách Sản phẩm dạng Text File:

```

fun main(args: Array<String>) {
    var data:MutableList<SanPham> = mutableListOf()
    var sp1=SanPham(1,"Coca cola",15.5)
    data.add(sp1)
    var sp2=SanPham(2,"Sting",25.0)
    data.add(sp2)
    var sp3=SanPham(3,"Redbull",17.0)
}

```

```

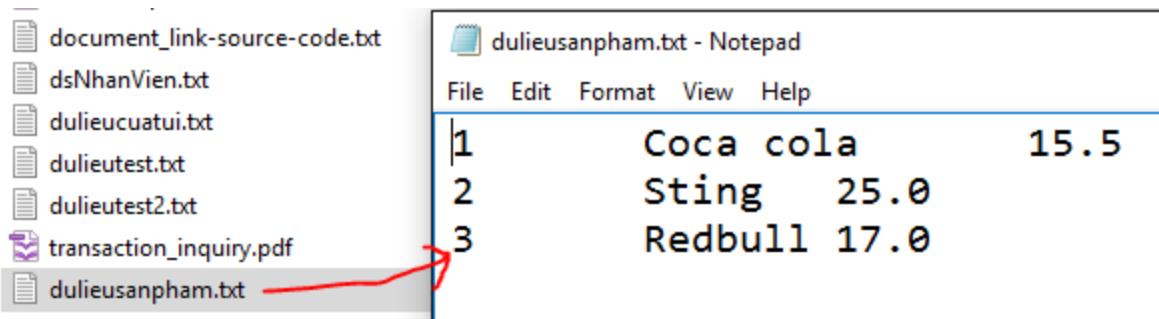
        data.add(sp3)
    var
kqLuu=TextFileFactory().LuuFile(data, "d:/dulieusanpham.txt")
    if(kqLuu)
    {
        println("Lưu text file thành công")
    }
    else
    {
        println("Lưu text file thất bại")
    }
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

Lưu text file thành công

Bây giờ ta vào ổ D xem tập tin dulieusanpham.txt có được lưu thành công hay chưa:



Rõ ràng kết quả đã lưu thành công, bây giờ ta sẽ gọi hàm đọc thông tin lên nhé:

```

fun main(args: Array<String>) {
    var data:MutableList<SanPham> =
TextFileFactory().DocFile("d:/dulieusanpham.txt")
    for (sp in data)
        println(sp)
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

1 Coca cola 15.5
2 Sting 25.0
3 Redbull 17.0

Như vậy ta đã lưu và đọc Text File thành công, các bạn tự áp dụng vào các dự án cụ thể nhé, Lưu cấu trúc Text File như thế nào là do quyết định của bạn, bài trên Tui lưu mỗi đối tượng là 1 dòng, và các thuộc tính ngăn cách bởi 1 dấu tab.

Các bài sau Tui sẽ trình bày về Xử lý Serialize File trong Kotlin rất quan trọng trong quá trình xử lý lưu trữ dữ liệu

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/tnyg7czel2zx7oy/HocTextFile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 30-Xử lý Serialize File trong Kotlin

Bạn đã được học xử lý Text File ở [bài 29](#), Trong bài này Tui tiếp tục hướng dẫn chuỗi bài học xử lý file, đó là Serialize File trong Kotlin

Cũng giống như Text File, Kotlin cũng dùng các thư viện JVM để xử lý Serialize nên nó cũng rất giống với Java.

Serialize File cho phép ta “chụp ảnh” đối tượng xuống ổ cứng và phục hồi hình ảnh từ ổ cứng lên bộ nhớ. Để lưu được dạng Serialize thì các lớp có lưu trữ xuống ổ cứng phải kế thừa interface **Serialize**.

Các gói thư viện dùng để chụp ảnh và phục hồi ảnh trong trường hợp này gồm:

```
import java.io.FileInputStream  
import java.io.FileOutputStream  
import java.io.ObjectInputStream  
import java.io.ObjectOutputStream
```

Ta tạo một Project mới, cho các lớp tương tự như bài 29 để các bạn dễ so sánh:

Lớp Sản phẩm sẽ implements interface Serialize như dưới đây:

```
import java.io.Serializable  
/*  
 * Created by cafe on 02/06/2017.  
 */  
class SanPham:Serializable {  
    var ma:Int=0  
    var ten:String=""  
    var donGia:Double=0.0  
    constructor()  
    constructor(ma: Int, ten: String, donGia: Double) {  
        this.ma = ma  
        this.ten = ten  
        this.donGia = donGia  
    }  
    override fun toString(): String {  
        return "$ma\t$ten\t$donGia"  
    }  
}
```

Tiếp tục tạo lớp **SerializableFileFactory** để cung cấp 2 hàm Lưu và đọc tập tin dạng Serialize, kỹ thuật viết như sau:

```
import java.io.FileInputStream
import java.io.FileOutputStream
import java.io.ObjectInputStream
import java.io.ObjectOutputStream

/**
 * Created by cafe on 02/06/2017.
 */
class SerializableFileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ
     * @return true nếu lưu thành công, false nếu lưu thất bại
     */
    fun LuuFile(data:MutableList<SanPham>, path:String):Boolean
    {
        try {
            var fos=FileOutputStream(path);
            var oos=ObjectOutputStream(fos);
            oos.writeObject(data);
            oos.close();
            fos.close();
            return true
        }
        catch (ex:Exception)
        {
            ex.printStackTrace()
        }
        return false
    }
    /**
     * @author Trần Duy Thành
     * @param path:đường dẫn muốn đọc dữ liệu
     * @return Danh sách sản phẩm MutableList
     */
    fun DocFile(path:String) :MutableList<SanPham>
    {
        var data:MutableList<SanPham> = mutableListOf()
        try
        {
            var fis=FileInputStream(path);
            var ois=ObjectInputStream(fis);
```

```

        var obj=ois.readObject();
        data= obj as MutableList<SanPham>;
        ois.close();
        fis.close();
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return data
}
}

```

Ta thấy cách lưu và đọc tập tin dạng Serialize đơn giản hơn rất nhiều so với Text File, và các bạn chú ý là nó không quan tâm cấu trúc mối quan hệ nhằng nhịt giữa các lớp như thế nào. Nó chụp 1 cái BUP là lưu hết toàn bộ xuống ổ cứng luôn.

Bây giờ tạo hàm main để Test lưu chụp ảnh đối tượng:

```

fun main(args: Array<String>) {
    var data:MutableList<SanPham> = mutableListOf()
    var sp1=SanPham(1, "Coca cola", 15.5)
    data.add(sp1)
    var sp2=SanPham(2, "Sting", 25.0)
    data.add(sp2)
    var sp3=SanPham(3, "Redbull", 17.0)
    data.add(sp3)
    var
    kqLuu=SerializableFileFactory().LuuFile(data, "d:/dulieusanpham.dat")
    if(kqLuu)
    {
        println("Lưu text file thành công")
    }
    else
    {
        println("Lưu text file thất bại")
    }
}

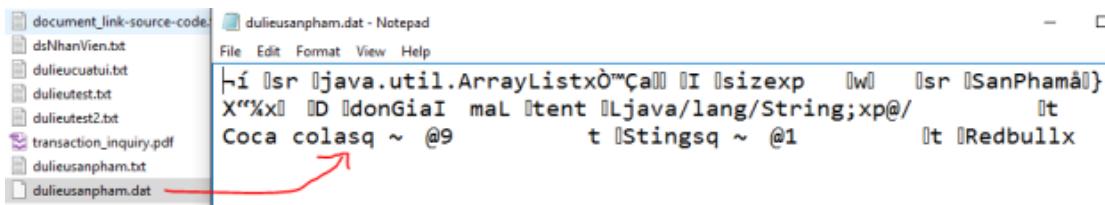
```

Ở trên Tui lưu tập tin với tập dulieusanpham.dat (đuôi gì cũng được):

Khi chạy hàm main ở trên thì ta có kết quả sau:

Lưu text file thành công

Bây giờ ta vào ô D xem tập tin dulieusanpham.dat có được lưu thành công hay chưa:



Mở bằng Notepad, bạn thấy nó ra giun dê loằng ngoằng, không giống như Text File

Rõ ràng kết quả đã lưu thành công, bây giờ ta sẽ gọi hàm đọc thông tin lên nhé:

```
fun main(args: Array<String>) {
    var data:MutableList<SanPham> =
SerializableFileFactory().DocFile("d:/dulieusanpham.dat")
    for (sp in data)
        println(sp)
}
```

Khi chạy hàm main ở trên thì ta có kết quả sau:

```
1 Coca cola 15.5
2 Sting 25.0
3 Redbull 17.0
```

Như vậy ta đã lưu và đọc Serialize File thành công, các bạn tự áp dụng vào các dự án cụ thể nhé, cấu trúc Serialize File là dạng nhị phân nên ta đọc không hiểu, chỉ phần mềm của ta đọc mới hiểu.

Các bài sau Tui sẽ trình bày về Xử lý XML File trong Kotlin rất quan trọng trong quá trình xử lý lưu trữ dữ liệu

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/v66y0alecg7q515/HocSerializeFile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 31-Xử lý XML File trong Kotlin

Chúng ta đã biết xử lý [Text File](#), [Serialize File](#), trong bài này Tui sẽ hướng dẫn các bạn cách lưu và đọc dữ liệu với XML File.

Tui vẫn sử dụng các thư viện trong JVM để xử lý cho Kotlin. Cũng với ví dụ như các bài xử lý file trước đó, ta có lớp Sản phẩm với thông tin như sau:

```
import java.io.Serializable

/**
 * Created by cafe on 02/06/2017.
 */
class SanPham {
    var ma:Int=0
    var ten:String=""
    var donGia:Double=0.0
    constructor()
    constructor(ma: Int, ten: String, donGia: Double) {
        this.ma = ma
        this.ten = ten
        this.donGia = donGia
    }
    override fun toString(): String {
        return "$ma\t$ten\t$donGia"
    }
}
```

Cấu trúc file XML Tui muốn các bạn phải lưu trữ thành:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SanPhams>
<SanPham>
<Ma>1</Ma>
<Ten>Coca cola</Ten>
<Gia>15.5</Gia>
</SanPham>
<SanPham>
<Ma>2</Ma>
<Ten>String</Ten>
<Gia>25.0</Gia>
</SanPham>
<SanPham>
<Ma>3</Ma>
```

```
<Ten>Redbull</Ten>
<Gia>17.0</Gia>
</SanPham>
</SanPhams>
```

Để lưu và đọc được XML File thì ta phải sử dụng các thư viện sau:

```
import java.io.File
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.transform.stream.StreamResult
import javax.xml.transform.dom.DOMSource
import javax.xml.transform.TransformerFactory
import org.w3c.dom.Element
```

Tui có tạo 1 lớp XMLFileFactory có 2 phương thức để ghi file XML và đọc file XML. Các bạn muốn hiểu rõ thêm về XML thì nên học thêm các kiến thức về XML. Trong bài học này Tui cung cấp các lệnh để các bạn có thể áp dụng vào việc ghi và đọc File (Tui không giải thích sâu, vì các bạn là dân lập trình nên chắc chắn đọc sẽ suy luận được. Nếu bạn khó hiểu thì cứ nhớ trong đầu như sau: Hàm LuuFile là hàm đưa dữ liệu danh sách Sản phẩm thành file XML, hàm DocFile mô hình hóa ngược lại từ tập dữ liệu XML thành hướng đối tượng trong Kotlin -là danh Sách Sản Phẩm).

```
import java.io.File
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.transform.stream.StreamResult
import javax.xml.transform.dom.DOMSource
import javax.xml.transform.TransformerFactory
import org.w3c.dom.Element

/**
 * Created by cafe on 02/06/2017.
 */
class XMLFileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ
     * @return true nếu lưu thành công, false nếu lưu thất bại
     */
    fun LuuFile(data:MutableList<SanPham>, path:String):Boolean
    {
        try
        {
            val docFactory =
```

```

DocumentBuilderFactory.newInstance()
    val docBuilder = docFactory.newDocumentBuilder()
// root elements
    val doc = docBuilder.newDocument()
    val rootElement = doc.createElement("SanPhams")
    doc.appendChild(rootElement)
    for(sp in data)
    {
        val sanPhamElement =
doc.createElement("SanPham")
        val maElement=doc.createElement("Ma")
        maElement.textContent=sp.ma.toString()
        sanPhamElement.appendChild(maElement)
        val tenElement=doc.createElement("Ten")
        tenElement.textContent=sp.ten
        sanPhamElement.appendChild(tenElement)
        val giaElement=doc.createElement("Gia")
        giaElement.textContent=sp.donGia.toString()
        sanPhamElement.appendChild(giaElement)
        rootElement.appendChild(sanPhamElement);
    }
// write the content into xml file
    val transformerFactory =
TransformerFactory.newInstance()
    val transformer =
transformerFactory.newTransformer()
        val source = DOMSource(doc)
        val result = StreamResult(File(path).absolutePath)

// Output to console for testing
// StreamResult result = new StreamResult(System.out);
        transformer.transform(source, result)
        return true
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return false
}
/***
 * @author Trần Duy Thành
 * @param path:đường dẫn muốn đọc dữ liệu
 * @return Danh sách sản phẩm MutableList
 */
fun DocFile(path:String) :MutableList<SanPham>
{

```

```

var data:MutableList<SanPham> = mutableListOf()
try {
//Get the DOM Builder Factory
    val factory = DocumentBuilderFactory.newInstance()
//Get the DOM Builder
    val builder = factory.newDocumentBuilder()
//Load and Parse the XML document
//document contains the complete XML as a Tree.
    val xmlfile = File(path)
    val document = builder.parse(xmlfile)
//Iterating through the nodes and extracting the data.
    val nodeList = document.documentElement.childNodes

    for (i in 0..nodeList.length - 1) {

//We have encountered an <SanPham> tag.
    val node = nodeList.item(i)
    if (node is Element) {
        val sp = SanPham()
        val childNodes = node.getChildNodes()
        for (j in 0..childNodes.getLength() - 1) {
            val cNode = childNodes.item(j)

//Identifying the child tag of employee encountered.
            if (cNode is Element) {
                val content =
cNode.getLastChild().gettextContent().trim()
                when (cNode.getNodeName()) {
                    "Ma" -> sp.ma= content.toInt()
                    "Ten" -> sp.ten= content
                    "Gia" -> sp.donGia=
content.toDouble()
                }
            }
            data.add(sp)
        }
    }
}
catch (ex:Exception)
{
    ex.printStackTrace()
}
return data
}
}

```

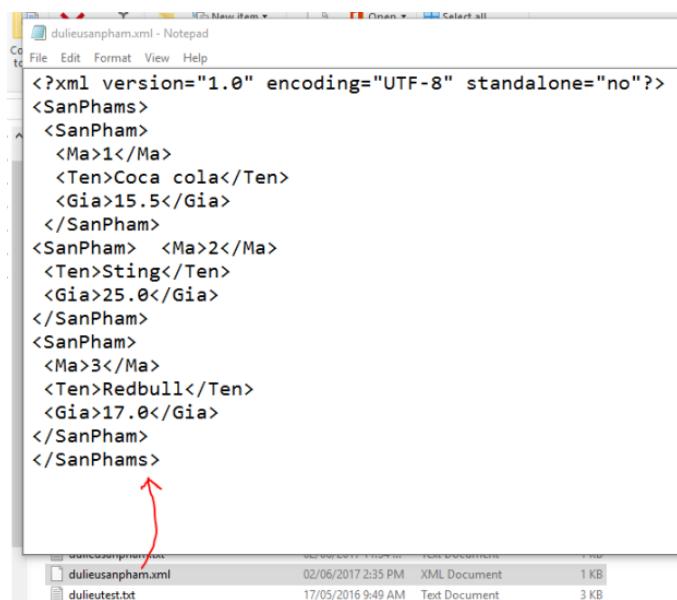
Ta tạo một hàm main để thử nghiệm việc lưu XML FILE:

```
fun main(args: Array<String>) {
    var data:MutableList<SanPham> = mutableListOf()
    var sp1=SanPham(1, "Coca cola", 15.5)
    data.add(sp1)
    var sp2=SanPham(2, "Sting", 25.0)
    data.add(sp2)
    var sp3=SanPham(3, "Redbull", 17.0)
    data.add(sp3)
    var kqLuu=
    XMLFileFactory().LuuFile(data, "d:/dulieusanpham.xml")
    if(kqLuu)
    {
        println("Lưu text file thành công")
    }
    else
    {
        println("Lưu text file thất bại")
    }
}
```

Khi chạy hàm main ở trên thì ta có kết quả sau:

Lưu text file thành công

Bây giờ ta vào ô D xem tập tin dulieusanpham.xml có được lưu thành công hay chưa:



Rõ ràng kết quả đã lưu thành công, bây giờ ta sẽ gọi hàm đọc thông tin lên nhé:

```
fun main(args: Array<String>) {
    var data:MutableList<SanPham> =
XMLFileFactory().DocFile("d:/dulieusanpham.xml")
    for (sp in data)
        println(sp)
}
```

Khi chạy hàm main ở trên thì ta có kết quả sau:

1 Coca cola 15.5
2 Sting 25.0
3 Redbull 17.0

Như vậy ta đã lưu và đọc XML File thành công, các bạn tự áp dụng vào các dự án cụ thể nhé, cấu trúc XML File như thế nào là do bạn quyết định

Các bài sau Tui sẽ trình bày về Xử lý JSON File trong Kotlin rất quan trọng trong quá trình xử lý lưu trữ dữ liệu

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/lh1024w1b722but/HocXMLFile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 32-Xử lý Json trong Kotlin – Bài 1

Các bạn đã nắm được 3 kiểu tương tác File: [Text File](#), [Serialize File](#), [XML file](#). Bây giờ Tui hướng dẫn loại định dạng file cuối cùng rất nổi tiếng hiện nay đó là định dạng Json.

Khái niệm về Json Tui đã trình bày ở [bài 51 của Android](#), các bạn có thể vào bài này để đọc thêm.

Kotlin cũng có sẵn các lớp để tương tác Json, hoặc có nhiều thư viện ngoài rất nổi tiếng như Gson, [Klaxon](#)... giúp chúng ta dễ dàng chuyển Object Model thành Json và từ Json thành Object Model vô cùng lợi hại.

Trong bài này Tui sẽ hướng dẫn các bạn cách dùng Gson trong Kotlin để chuyển đổi qua loại giữa Object Model và Json.

Tính tới thời điểm Tui viết bài học này thì Gson có phiên bản mới nhất là 2.8.1 (update ngày 31/05/2017)

Bạn cần tải thư viện này về rồi reference nó vào Project trong IntelliJ IDEA của bạn.

Để tải Gson 2.8.1 bạn vào link:

<https://repo1.maven.org/maven2/com/google/code/gson/2.8.1/>

Tải tập tin [gson-2.8.1.jar](#) (dung lượng khoảng 228kb) từ link ở trên về máy tính.

Tui nói sơ qua cách thức hoạt động của Gson:

Để chuyển đổi Kotlin Model tới Json ta làm như sau:

```
val gson = Gson()
val obj = KotlinModel() //lớp nào đó trong Kotlin

// 1. Kotlin object to JSON, and save into a file
val file=FileWriter("D:\\file.json")
gson.toJson(obj, file)
file.close()

// 2. Kotlin object to JSON, and assign to a String
val jsonInString = gson.toJson(obj)
```

Để chuyển đổi JSON về Kotlin Model ta làm như sau:

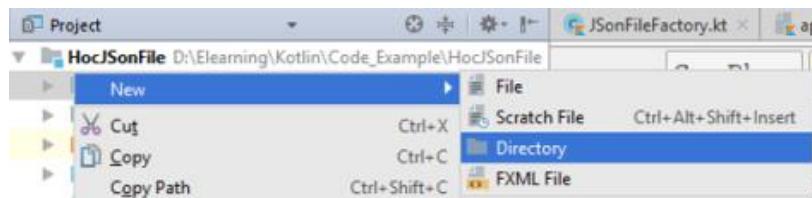
```
val gson = Gson()
// 1. JSON to Kotlin model, read it from a file.
val data = gson.fromJson(FileReader("D:\\file.json"),
SanPham::class.java)
// 2. JSON to Kotlin Model, read it from a Json String.
val jsonInString = "{\"name\" : \"cocacola\"}"
val data= gson.fromJson(jsonInString, SanPham::class.java)
// JSON to JsonElement, convert to String later.
val json = gson.fromJson(FileReader("D:\\file.json"),
JsonElement::class.java)
val result = gson.toJson(json)
```

Nếu dữ liệu Json là dạng JSONArray thì để đưa về Kotlin Model ta phải làm như sau:

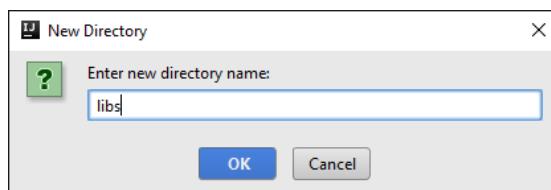
```
val gson = Gson()
val json = "[{\"name\":\"cocacola\"}, {"name\":\"pepsi\"}]"
val data:MutableList = gson.fromJson(json,
object : TypeToken()
{} .type)
```

Bây giờ Tui sẽ hướng dẫn chi tiết từng bước cụ thể để các bạn có thể dễ dàng hiểu và vận dụng thư viện GSon.

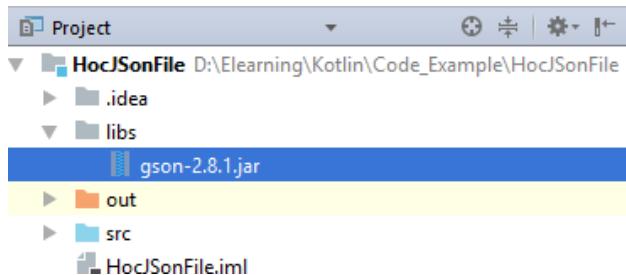
Tạo một Project tên là HocJsonFile, từ Project này ta tạo 1 thư mục(directory) tên là **libs** để chép thư viện **gson-2.8.1.jar** vào **libs**:



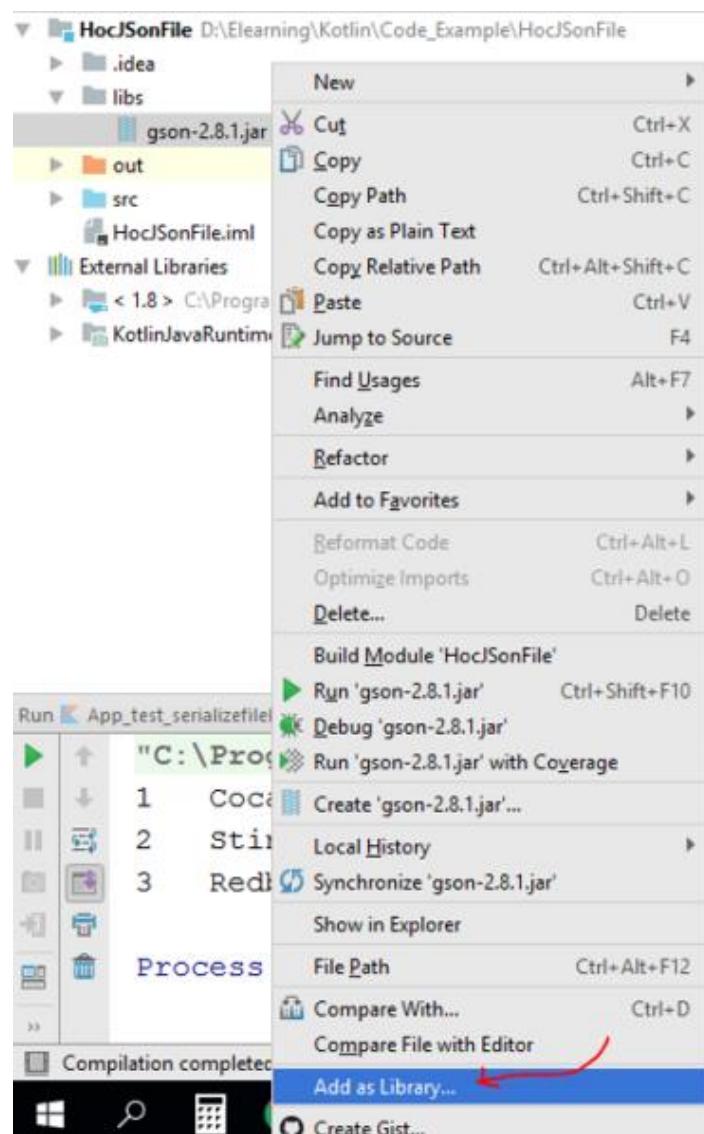
Bấm chuột phải vào Project/ chọn New/ chọn Directory:



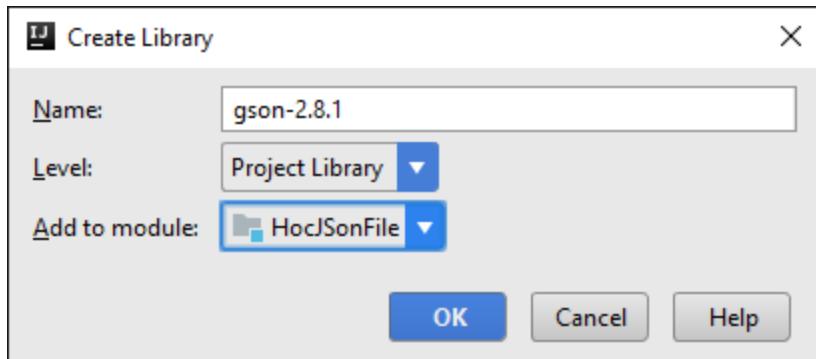
Đặt tên directory là libs rồi bấm Ok, lúc này thư mục **libs** sẽ được tạo ra trong Project. Ta chép [gson-2.8.1.jar](#) vào thư mục này như hình dưới đây:



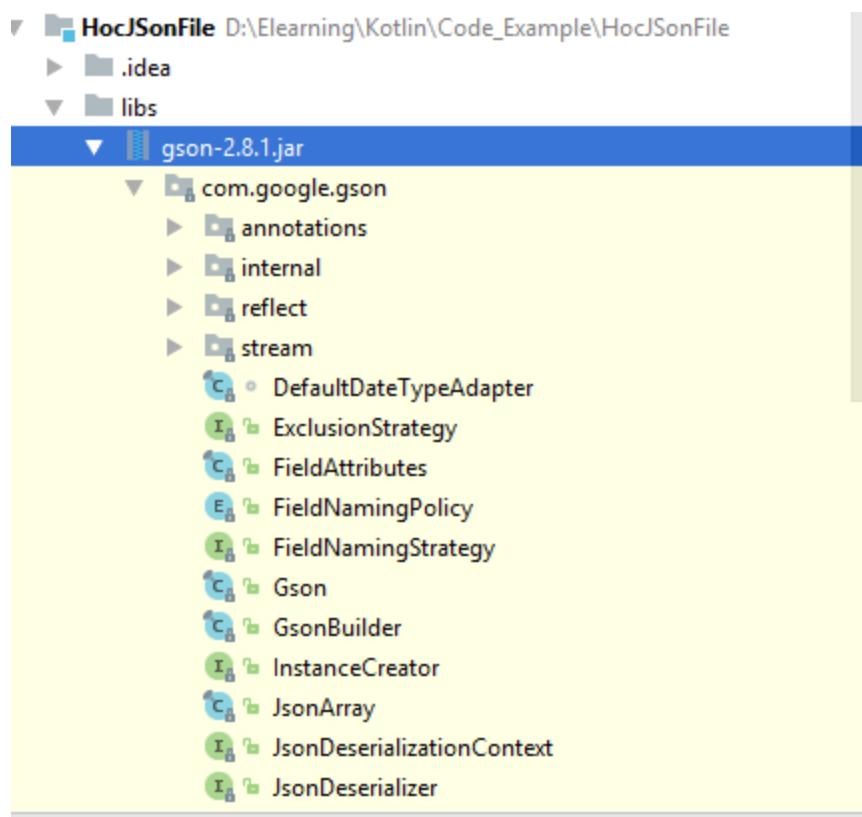
Tiếp theo ta cần đưa gson-2.8.1.jar thành thư viện sử dụng trong Project, cách làm như sau:



Bấm chuột phải vào gson-2.8.1.jar rồi chọn **add as Library** như hình ở trên, màn hình cấu hình sẽ xuất hiện:



Đặt tên(để mặc định) rồi bấm OK, lúc này bạn quan sát có sự thay đổi trong cách hiển thị, thấy được tập các lớp nằm trong thư viện này:



Bây giờ ta tiến hành tạo lớp Sản phẩm giống như các kỹ thuật xử lý text file, serialize file, xml file mà bạn đã được học trước đó:

```

import java.io.Serializable
/**
 * Created by cafe on 02/06/2017.
 */
class SanPham:Serializable {
    var ma:Int=0
    var ten:String=""
    var donGia:Double=0.0
    constructor()
    constructor(ma: Int, ten: String, donGia: Double) {
        this.ma = ma
        this.ten = ten
        this.donGia = donGia
    }
    override fun toString(): String {
        return "$ma\t$ten\t$donGia"
    }
}

```

Tiếp tục tạo lớp JSONFileFactory để Lưu và đọc Json bằng Gson:

```

import com.google.gson.Gson
import java.io.FileWriter
import java.io.FileReader
import com.google.gson.reflect.TypeToken

/**
 * Created by cafe on 02/06/2017.
 */
class JSONFileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ
     * @return true nếu lưu thành công, false nếu lưu thất bại
     */
    fun LuuFile(data:MutableList<SanPham>,path:String):Boolean
    {
        try {
            val gs= Gson()
            val file=FileWriter(path)
            gs.toJson(data,file)
            file.close()
            return true
        }
        catch (ex:Exception)
        {

```

```

        ex.printStackTrace()
    }
    return false
}
/***
 * @author Trần Duy Thành
 * @param path:đường dẫn muốn đọc dữ liệu
 * @return Danh sách sản phẩm MutableList
 */
fun DocFile(path:String) :MutableList<SanPham>
{
    var data:MutableList<SanPham> = mutableListOf()
    try
    {
        val gson = Gson()
        var file=FileReader(path)
        data = gson.fromJson<MutableList<SanPham>>(file,
            object : TypeToken<MutableList<SanPham>>()
            {
                .
            }.type
        )
        file.close()
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return data
}
}

```

Bạn thấy đây, Gson giúp chúng ta đơn giản hóa mọi việc lưu và đọc dữ liệu. Đây là một trong những thư viện nổi tiếng, được sử dụng rất nhiều trong các dự án, và định dạng dữ liệu JSON ngày càng phổ biến, có thể hơn cả XML vốn đã nổi đình đám trước đó.

Cuối cùng ta tạo hàm main để kiểm tra:

```

fun main(args: Array<String>) {

    var data:MutableList<SanPham> = mutableListOf()
    var sp1=SanPham(1, "Coca cola", 15.5)
    data.add(sp1)
    var sp2=SanPham(2, "Sting", 25.0)
    data.add(sp2)
    var sp3=SanPham(3, "Redbull", 17.0)
    data.add(sp3)
}

```

```

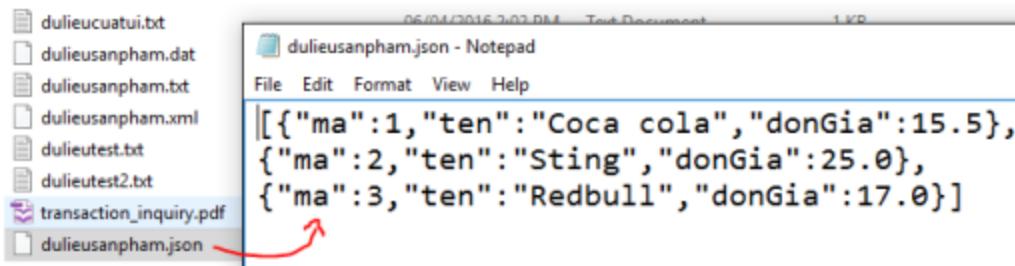
var kqLuu=
JSONFileFactory().LuuFile(data, "d:/dulieusanpham.json")
if(kqLuu)
{
    println("Lưu Json file thành công")
}
else
{
    println("Lưu Json file thất bại")
}
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

Lưu Json file thành công

Bây giờ ta vào ô D xem tập tin **dulieusanpham.json** có được lưu thành công hay chưa:



Rõ ràng kết quả đã lưu thành công, bây giờ ta sẽ gọi hàm đọc thông tin lên nhé:

```

fun main(args: Array<String>) {
    var data:MutableList<SanPham> =
JSONFileFactory().DocFile("d:/dulieusanpham.json")
    for (sp in data)
        println(sp)
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

1 Coca cola 15.5
2 Sting 25.0
3 Redbull 17.0

Như vậy ta đã lưu và đọc JSON File thành công, các bạn tự áp dụng vào các dự án cũ thênhé, cấu trúc JSON File trong trường hợp này nó sẽ tự động build dựa vào cấu trúc Class và mối quan hệ giữa các Class mà bạn tạo ra.

Bài sau Tui sẽ trình bày thêm cách lấy dữ liệu Json từ Internet trong Kotlin, các Json có cấu trúc phức tạp, các bạn chú ý theo dõi nhé

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/7jgvvzyschy77gg/HocJSONFile.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

Bài 33-Xử lý Json trong Kotlin – Bài 2

Trong [bài 32](#) Tui đã trình bày chi tiết cách sử dụng thư viện GSon để Lưu Kotlin Model thành JSON và Đọc JSON thành Kotlin Model như thế nào. Ở bài này Tui tiếp tục làm thêm một ví dụ phức tạp hơn về JSON trong Kotlin, đó là tạo ra 2 lớp có mối quan hệ Master-Detail, đây là một trong những trường hợp thường gặp nhiều nhất trong quá trình triển khai dự án thật. Cụ thể Tui sẽ bổ sung thêm một Lớp Danh Mục, nó có mối quan hệ với Lớp Sản Phẩm: Một Danh Mục có nhiều Sản phẩm và Một Sản phẩm thuộc về một danh Mục nào đó. Để qua đây chúng ta tìm hiểu xem GSon tạo ra file JSON như thế nào cũng như phục hồi lại Kotlin Model ra sao.

Bài này Tui sẽ đi trực tiếp vào kỹ thuật lập trình luôn, còn lý thuyết các bạn tự xem lại [bài 32](#) nhé.

Lớp Sản Phẩm có cấu trúc như sau:

```
import java.io.Serializable

/**
 * Created by cafe on 03/06/2017.
 */
class SanPham {
    var MaSanPham:Int=0
    var TenSanPham:String=""
    var DonGia:Double=0.0
    constructor()
    constructor(MaSanPham: Int, TenSanPham: String, DonGia: Double) {
        this.MaSanPham = MaSanPham
        this.TenSanPham = TenSanPham
        this.DonGia = DonGia
    }
    override fun toString(): String {
        return MaSanPham.toString()+"\t"+TenSanPham+"\t"+DonGia
    }
}
```

Cấu trúc của lớp Danh Mục:

```
import java.io.Serializable

/**
```

```

 * Created by cafe on 03/06/2017.
 */
class DanhMuc {
    var MaDanhMuc:Int=0
    var TenDanhMuc:String=""
    var SanPhams:MutableList<SanPham> = mutableListOf()
    constructor()
    constructor(MaDanhMuc: Int, TenDanhMuc: String) {
        this.MaDanhMuc = MaDanhMuc
        this.TenDanhMuc = TenDanhMuc
    }
    override fun toString(): String {
        var s=""
        for (sp in SanPhams)
            s+="\t"+sp.toString() + "\n"
        var infor="Danh Mục: ["+MaDanhMuc.toString()+
"\t"+TenDanhMuc+"]"
        infor+="\nCác Sản phẩm của danh Mục này là:\n"+s
        return infor
    }
    fun ThemSanPham(sp:SanPham)
    {
        SanPhams.add(sp)
    }
}

```

Như Tui đã nói ở bài 32, GSon không quan tâm cấu trúc và mối quan hệ giữa các lớp mà bạn viết như thế nào. Nó “cân” hết (đừng bị đệ quy là được, nếu ko nó bị stack overflow)

Giờ ta tiếp tục tạo lớp JSONFileFactory, nó có cấu trúc như dưới đây (giống nhau trong mọi trường hợp nếu bạn dùng kiểu dữ liệu là Any)

```

import com.google.gson.Gson
import java.io.FileWriter
import java.io.FileReader
import com.google.gson.reflect.TypeToken

/**
 * Created by cafe on 02/06/2017.
 */
class JSONFileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ

```

```

    * @return true nếu lưu thành công, false nếu lưu thất bại
    */
fun LuuFile(data:MutableList<DanhMuc>, path:String):Boolean
{
    try {
        val gs= Gson()
        val file=FileWriter(path)
        gs.toJson(data,file)
        file.close()
        return true
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return false
}
/**
 * @author Trần Duy Thành
 * @param path:đường dẫn muốn đọc dữ liệu
 * @return Danh sách sản phẩm MutableList
 */
fun DocFile(path:String) :MutableList<DanhMuc>
{
    var data:MutableList<DanhMuc> = mutableListOf()
    try
    {
        val gson = Gson()
        var file=FileReader(path)
        data = gson.fromJson<MutableList<DanhMuc>>(file,
            object : TypeToken<MutableList<DanhMuc>>()
            {
                .
            }.type
        )
        file.close()
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return data
}
}

```

Cuối cùng ta tạo hàm main để kiểm tra:

```

fun main(args: Array<String>) {
    var database:MutableList<DanhMuc> = mutableListOf<DanhMuc>()

    var dmDienTu:DanhMuc=DanhMuc(1, "Mặt hàng điện tử")
    database.add(dmDienTu)

    var bongden:SanPham=SanPham(1, "Bóng đèn điện Quang", 150.0)
    dmDienTu.ThemSanPham(bongden)
    var acquy:SanPham=SanPham(2, "Ác quy Đồng Nai", 250.0)
    dmDienTu.ThemSanPham(acquy)
    var maydien:SanPham=SanPham(3, "Máy phát điện ABC", 90.0)
    dmDienTu.ThemSanPham(maydien)

    var dmTieuDung:DanhMuc=DanhMuc(2, "Mặt hàng tiêu dùng")
    database.add(dmTieuDung)

    var xabong:SanPham=SanPham(4, "Xà Bông Lifeboy", 15.0)
    dmTieuDung.ThemSanPham(xabong)
    var nuocruachen:SanPham=SanPham(5, "Nước rửa chén
Sunlight", 12.0)
    dmTieuDung.ThemSanPham(nuocruachen)

    var dmHoaChat:DanhMuc=DanhMuc(3, "Mặt hàng Hóa Chất")
    database.add(dmHoaChat)

    var dietmuoi:SanPham=SanPham(6, "Thuốc Diệt Muỗi XYZ", 80.0)
    dmHoaChat.ThemSanPham(dietmuoi)
    var dietchuot:SanPham=SanPham(7, "Thuốc Diệt Chuỗi ABC", 70.0)
    dmHoaChat.ThemSanPham(dietchuot)

    for (dm in database)
        println(dm)

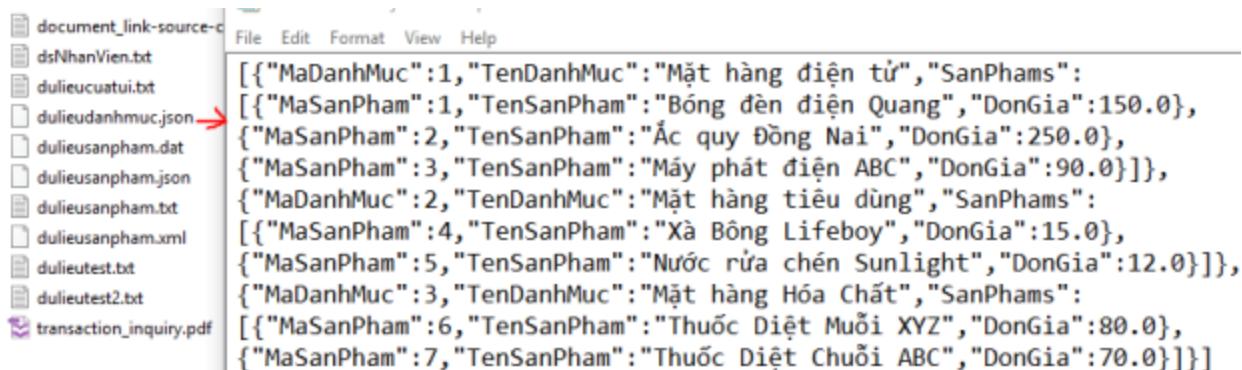
    var kqLuu=
    JJsonFileFactory().LuuFile(database, "d:/dulieudanhmuc.json")
    if(kqLuu)
    {
        println("Lưu Json file thành công")
    }
    else
    {
        println("Lưu Json file thất bại")
    }
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

```
Danh Mục: [1 Mặt hàng điện tử]
Các Sản phẩm của danh Mục này là:
1 Bóng đèn điện Quang 150.0
2 Ấc quy Đồng Nai 250.0
3 Máy phát điện ABC 90.0
Danh Mục: [2 Mặt hàng tiêu dùng]
Các Sản phẩm của danh Mục này là:
4 Xà Bông Lifeboy 15.0
5 Nước rửa chén Sunlight 12.0
Danh Mục: [3 Mặt hàng Hóa Chất]
Các Sản phẩm của danh Mục này là:
6 Thuốc Diệt Muỗi XYZ 80.0
7 Thuốc Diệt Chuỗi ABC 70.0
Lưu Json file thành công
```

Bây giờ ta vào ô D xem tập tin **dulieudanhmuc.json** có được lưu thành công hay chưa:



Rõ ràng kết quả đã lưu thành công, bạn quan sát thấy cấu trúc Json này có gì khác biệt? Đó là mỗi một đối tượng danh Mục nó một mảng SanPhams → được GSon tự động tạo ra từ mối quan hệ của 2 Lớp DanhMuc + SanPham

Bây giờ ta sẽ gọi hàm đọc thông tin lên nhé:

```
fun main(args: Array<String>) {
    var database:MutableList<DanhMuc> =
        JSONFileFactory().DocFile("d:/dulieudanhmuc.json")
    for (dm in database)
        println(dm)
}
```

Khi chạy hàm main ở trên thì ta có kết quả sau:

Danh Mục: [1 Mặt hàng điện tử]

Các Sản phẩm của danh Mục này là:

1 Bóng đèn điện Quang 150.0

2 Ấc quy Đồng Nai 250.0

3 Máy phát điện ABC 90.0

Danh Mục: [2 Mặt hàng tiêu dùng]

Các Sản phẩm của danh Mục này là:

4 Xà Bông Lifeboy 15.0

5 Nước rửa chén Sunlight 12.0

Danh Mục: [3 Mặt hàng Hóa Chất]

Các Sản phẩm của danh Mục này là:

6 Thuốc Diệt Muỗi XYZ 80.0

7 Thuốc Diệt Chuỗi ABC 70.0

Có vẻ tới đây các bạn thấy rằng JSON sẽ dễ lập trình hơn Text File, Serialize File, và XML File đúng không? Hiện nay cấu trúc JSON được sử dụng rất nhiều, ngày càng phổ biến và các lập trình viên rất thích điều này.

Như vậy ta đã Ví dụ xong trường hợp phức tạp của JSON File đó là có mối quan hệ giữa các lớp, các bạn tự áp dụng vào các dự án cụ thể nhé. Bài sau Tui sẽ trình bày thêm cách lấy dữ liệu JSON từ Internet trong Kotlin. Cụ thể là lấy Tỉ Giá hối đoái của Ngân Hàng Đông Á, JSON có cấu trúc phức tạp, các bạn chú ý theo dõi nhé

Các bạn có thể tải source code bài này ở đây:

http://www.mediafire.com/file/hk795a31148wsb0/HocJSON_DanhMucSanPham.rar

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 34-Đọc JSON tι giá hối đoái của Ngân Hàng Đông Á trong Kotlin – Bài 3

Tui đã trình bày kỹ cách sử dụng JSON trong Kotlin ở các [bài 32](#), bài 33 . Tui muốn minh họa thêm một ví dụ cuối cùng về đọc JSON trong Kotlin, trong bài này Tui sẽ coding lấy Tι giá hối đoái của Ngân Hàng Đông Á cung cấp:

bạn vào link này để xem Webservice API cung Tι giá hối đoái của DongA Bank:
<http://dongabank.com.vn/exchange/export>

Đây là kết quả (nó luôn chính xác và thời gian thực nhé các bạn, hàng ngày Ngân hàng Đông Á luôn cập nhật giá khi có sự thay đổi):

```
({"items": [{"type": "CAD", "imageurl": "http://www.dongabank.com.vn/images/flag/CAD.gif", "muatienmat": "16720", "muack": "16740", "bantienmat": "16930", "banck": "16930"}, {"type": "XAU", "imageurl": "http://www.dongabank.com.vn/images/flag/XAU.gif", "muatienmat": "3632000", "muack": "3621000", "bantienmat": "3655000", "banck": "3621000"}, {"type": "AUD", "imageurl": "http://www.dongabank.com.vn/images/flag/AUD.gif", "muatienmat": "16790", "muack": "16800", "bantienmat": "16980", "banck": "17000"}, {"type": "CHF", "imageurl": "http://www.dongabank.com.vn/images/flag/CHF.gif", "muatienmat": "23430", "muack": "23450", "bantienmat": "23730", "banck": "23730"}, {"type": "CNY", "imageurl": "http://www.dongabank.com.vn/images/flag/CNY.gif", "muatienmat": "3000", "muack": "3000", "bantienmat": "3500", "banck": "3500"}, {"type": "EUR", "imageurl": "http://www.dongabank.com.vn/images/flag/EUR.gif", "muatienmat": "25460", "muack": "25480", "bantienmat": "25750", "banck": "25750"}, {"type": "GBP", "imageurl": "http://www.dongabank.com.vn/images/flag/GBP.gif", "muatienmat": "29070", "muack": "29110", "bantienmat": "29450", "banck": "29450"}, {"type": "HKD", "imageurl": "http://www.dongabank.com.vn/images/flag/HKD.gif", "muatienmat": "2550", "muack": "2900", "bantienmat": "2940", "banck": "2940"}, {"type": "JPY", "imageurl": "http://www.dongabank.com.vn/images/flag/JPY.gif", "muatienmat": "204.3", "muack": "204.6", "bantienmat": "206.7", "banck": "206.7"}, {"type": "NZD", "imageurl": "http://www.dongabank.com.vn/images/flag/NZD.gif", "muatienmat": "16120", "muack": "16320", "bantienmat": "16320"}, {"type": "PNJ_DAB", "imageurl": "http://www.dongabank.com.vn/images/flag/PNJ_DAB.gif", "muatienmat": "3495000", "muack": "3495000", "bantienmat": "3545000", "banck": "3545000"}, {"type": "SGD", "imageurl": "http://www.dongabank.com.vn/images/flag/SGD.gif", "muatienmat": "16320", "muack": "16340", "bantienmat": "16540", "banck": "16540"}, {"type": "THB", "imageurl": "http://www.dongabank.com.vn/images/flag/THB.gif", "muatienmat": "595", "muack": "625", "bantienmat": "655", "banck": "655"}, {"type": "USD", "imageurl": "http://www.dongabank.com.vn/images/flag/USD.gif", "muatienmat": "22680", "muack": "22680", "bantienmat": "22750", "banck": "22750"}]})
```

Nhu chúng ta đã học ở những bài lý thuyết Json thì đây không phải là cấu trúc Json. Vì cấu trúc JSON chỉ chấp nhận 2 loại đó là: Dữ liệu được để trong {} hoặc được để trong [] .

Còn ở đây Ngân Hàng Đông Á lại để() ở ngoài. Do đó lúc đọc dữ liệu về ta chỉ cần Remove 2 ký tự này đi là xong.

Tiếp tục phân tích cấu trúc của Json ở trên:

```
object {1}
  array {1}
    items [14]
      0 {6}
        type : CAD
        imageurl : http://www.dongabank.com.vn/images\\flag\\CAD.gif
        muatienmat : 16720
        muack : 16740
        bantienmat : 16930
        banck : 16930
      1 {6}
        type : XAU
        imageurl : http://www.dongabank.com.vn/images\\flag\\XAU.gif
        muatienmat : 3632000
        muack : 3621000
        bantienmat : 3655000
        banck : 3621000
      2 {6}
```

Rõ ràng ở ngoài là 1 đối tượng JsonObject, bên trong nó có một JSONArray đặt tên là **items**. Mỗi đối tượng trong mảng Items có 6 thuộc tính như ở trên.

Bây giờ ta tạo một Project, và đưa thư viện gson-2.8.1.jar như các bài trước vào (bạn tự làm vì Tui đã hướng dẫn rất chi tiết ở các bài trước), rồi tiến hành tạo các lớp sau:

Lớp Item để lưu trữ các dữ liệu bên trong mảng items (chú ý là các thuộc tính phải copy paste y xì từ dữ liệuJson mà Đông Á Bank cung cấp):

```
class Item {
  var type:String=""
  var imageurl:String=""
  var muatienmat:String=""
```

```

var muack:String=""
var bantienmat:String=""
var banck:String=""
override fun toString(): String {
    return "Mã Tiền Tệ : "+type+"\n"+
        "Mua tiền mặt :" +muatienmat+"\n"+
        "Bán tiền mặt :" +bantienmat+"\n"+
        "Mua chuyển khoản :" +muack+"\n"+
        "Bán chuyển khoản :" +banck+"\n"+
        "Hình đại diện :" +imageurl+"\n"
}
}

```

Tiếp theo tạo 1 Lớp tỉ giá để lưu trữ mảng items

Các bạn cần nhớ là với GSon nó không quan tâm tên Lớp là gì (đặt tên gì cũng được), nó quan tâm tên thuộc tính (phải đặt chính xác như dữ liệu Json cũng cấp)

```

/**
 * Created by cafe on 03/06/2017.
 */
class TiGia {
    var items:MutableList<Item> = mutableListOf()
}

```

Cuối cùng tạo hàm main để kiểm tra lấy dữ liệu từ Ngân Hàng Đông Á, chú ý ta dùng HttpURLConnection để lấy dữ liệu từ Webservice API mà Đông Á Cung cấp.:

```

import com.google.gson.Gson
import java.io.BufferedReader
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL

/**
 * Created by cafe on 03/06/2017.
 */
fun main(args: Array<String>) {
    var url:URL=URL("http://dongabank.com.vn/exchange/export")
    var connection:HttpURLConnection= url.openConnection() as
HttpURLConnection
    var isr:InputStreamReader=
InputStreamReader(connection.getInputStream, "UTF-8")
    var br:BufferedReader=BufferedReader(isr)
    var s= br.readText()
}

```

```

        br.close()
        isr.close()
        s=s.replace("(", "")
        s=s.replace(")", "")
        var gs:Gson= Gson()
        var dstg:TiGia=gs.fromJson<TiGia>(s,TiGia::class.java)
        for (item in dstg.items)
        {
            println(item)
            println("-----")
        }
    }
}

```

Khi chạy hàm main ở trên thì ta có kết quả sau:

Mã Tiền tệ : CAD
 Mua tiền mặt :16720
 Bán tiền mặt :16930
 Mua chuyển khoản :16740
 Bán chuyển khoản :16930
 Hình đại diện :<http://www.dongabank.com.vn/images/flag/CAD.gif>

Mã Tiền tệ : XAU
 Mua tiền mặt :3632000
 Bán tiền mặt :3655000
 Mua chuyển khoản :3621000
 Bán chuyển khoản :3621000
 Hình đại diện :<http://www.dongabank.com.vn/images/flag/XAU.gif>

Mã Tiền tệ : AUD
 Mua tiền mặt :16790
 Bán tiền mặt :16980
 Mua chuyển khoản :16800
 Bán chuyển khoản :17000
 Hình đại diện :<http://www.dongabank.com.vn/images/flag/AUD.gif>

Mã Tiền tệ : CHF
 Mua tiền mặt :23430
 Bán tiền mặt :23730
 Mua chuyển khoản :23450
 Bán chuyển khoản :23730
 Hình đại diện :<http://www.dongabank.com.vn/images/flag/CHF.gif>

Mã Tiền tệ : CNY
 Mua tiền mặt :3000
 Bán tiền mặt :3500
 Mua chuyển khoản :3000

Bán chuyển khoản :3500

Hình đại diện :<http://www.dongabank.com.vn/images/flag/CNY.gif>

Mã Tiền tệ : EUR

Mua tiền mặt :25460

Bán tiền mặt :25750

Mua chuyển khoản :25480

Bán chuyển khoản :25750

Hình đại diện :<http://www.dongabank.com.vn/images/flag/EUR.gif>

Mã Tiền tệ : GBP

Mua tiền mặt :29070

Bán tiền mặt :29450

Mua chuyển khoản :29110

Bán chuyển khoản :29450

Hình đại diện :<http://www.dongabank.com.vn/images/flag/GBP.gif>

Mã Tiền tệ : HKD

Mua tiền mặt :2550

Bán tiền mặt :2940

Mua chuyển khoản :2900

Bán chuyển khoản :2940

Hình đại diện :<http://www.dongabank.com.vn/images/flag/HKD.gif>

Mã Tiền tệ : JPY

Mua tiền mặt :204.3

Bán tiền mặt :206.7

Mua chuyển khoản :204.6

Bán chuyển khoản :206.7

Hình đại diện :<http://www.dongabank.com.vn/images/flag/JPY.gif>

Mã Tiền tệ : NZD

Mua tiền mặt :

Bán tiền mặt :

Mua chuyển khoản :16120

Bán chuyển khoản :16320

Hình đại diện :<http://www.dongabank.com.vn/images/flag/NZD.gif>

Mã Tiền tệ : PNJ_DAB

Mua tiền mặt :3495000

Bán tiền mặt :3545000

Mua chuyển khoản :3495000

Bán chuyển khoản :3545000

Hình đại diện :http://www.dongabank.com.vn/images/flag/PNJ_DAB.gif

Mã Tiền tệ : SGD

Mua tiền mặt :16320
Bán tiền mặt :16540
Mua chuyển khoản :16340
Bán chuyển khoản :16540
Hình đại diện :<http://www.dongabank.com.vn/images/flag/SGD.gif>

Mã Tiền Tệ : THB
Mua tiền mặt :595
Bán tiền mặt :655
Mua chuyển khoản :625
Bán chuyển khoản :655
Hình đại diện :<http://www.dongabank.com.vn/images/flag/THB.gif>

Mã Tiền Tệ : USD
Mua tiền mặt :22680
Bán tiền mặt :22750
Mua chuyển khoản :22680
Bán chuyển khoản :22750
Hình đại diện :<http://www.dongabank.com.vn/images/flag/USD.gif>

Như vậy ta đã Ví dụ xong trường hợp đọc JSON từ 1 Webservice, cụ thể là Tỉ giá hối đoái của Ngân Hàng Đông Á, các bạn tự áp dụng vào các dự án cụ thể nhé. Các bài sau Tui sẽ hướng dẫn thiết kế giao diện (GUI) trong Kotlin, các bạn chú ý theo dõi nhé

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/hghudhgyo53ra8g/HocJsonDongABank.rar>

Hẹn gặp các bạn ở những bài tiếp theo

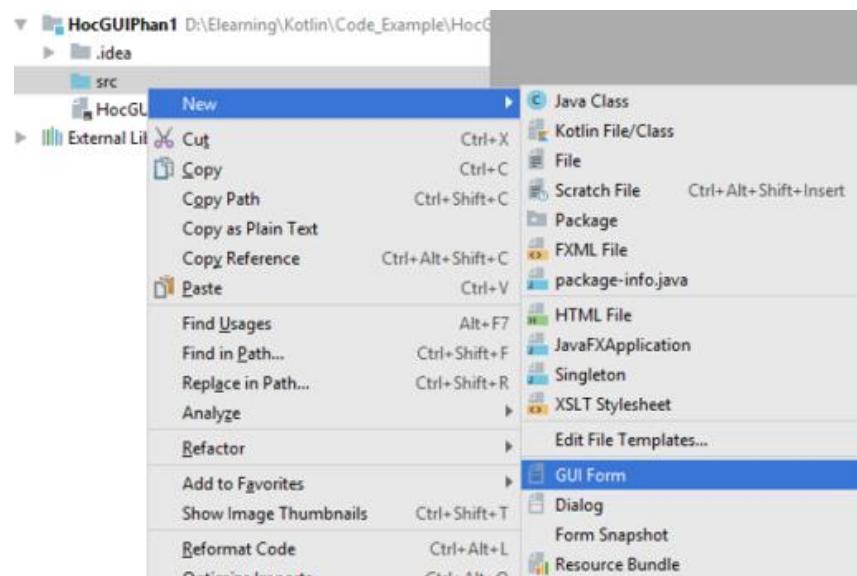
Chúc các bạn thành công!

Trần Duy Thanh (<http://ssoftinc.com/>)

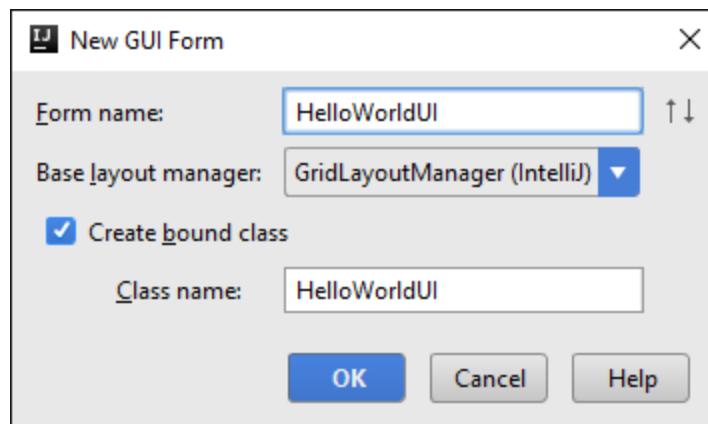
Bài 35-Thiết kế giao diện trong Kotlin – phần 1

Tính tới thời điểm 03/06/2017 thì Kotlin hiện tại chưa phát triển GUI Framework riêng mà đang sử dụng JVM để thiết kế giao diện (dùng awt, swing, javafx). Công cụ IntelliJ IDEA có hỗ trợ Drag & Drop giúp Lập Trình Viên có thể kéo thả thiết kế giao diện lúc Design Time.

Ta tạo một Projec tên là [HocGUIPhan1](#), sau đó New một GUI Form như cách sau:

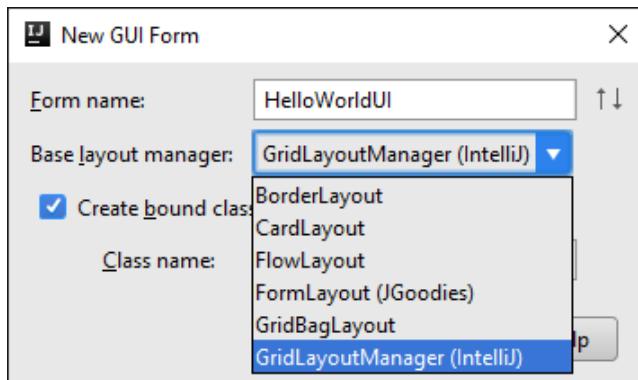


Bấm chuột phải vào Project/ chọn New / rồi chọn GUI Form, màn hình New GUI Form sẽ xuất hiện như dưới đây:



Form Name: Đặt tên cho Lớp giao diện, ví dụ HelloWorldUI

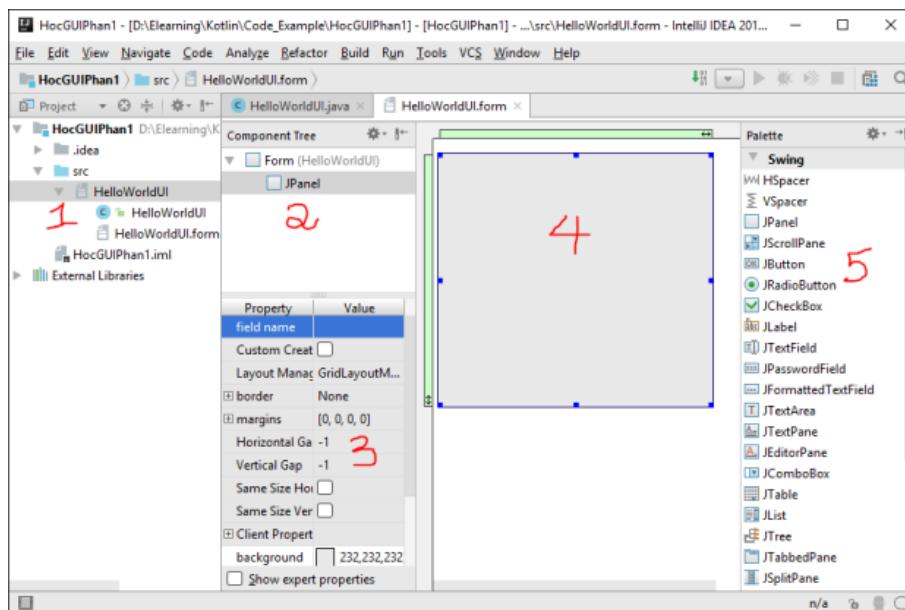
Base layout manager: Chọn layout mặc định cho giao diện, dĩ nhiên ta có thể thay đổi sau đó. Layout chính là nơi tổ chức sắp xếp các Control trên giao diện. Tại màn hình này, IntelliJ IDEA cung cấp một số Layout sau:



Vì lần đầu thử nghiệm, cứ để mặc định như IDEA chọn. Và Tui cũng không đi sâu vào giải thích từng thành phần, vì nó có sẵn trong Java Swing, [các bạn muốn chi tiết thì đăng ký học tại đây](#):

Create bound class: Nên checked để nó tự động tạo ra lớp tương tác giao diện.

Sau đó bạn nhấn OK để tạo, giao diện sẽ như hình dưới đây:



Ở màn hình trên Tui đánh dấu 5 phần, mỗi phần có những chức năng và ý nghĩa khác nhau:

Mục 1: Là cấu trúc tổ chức các Lớp, thư viện

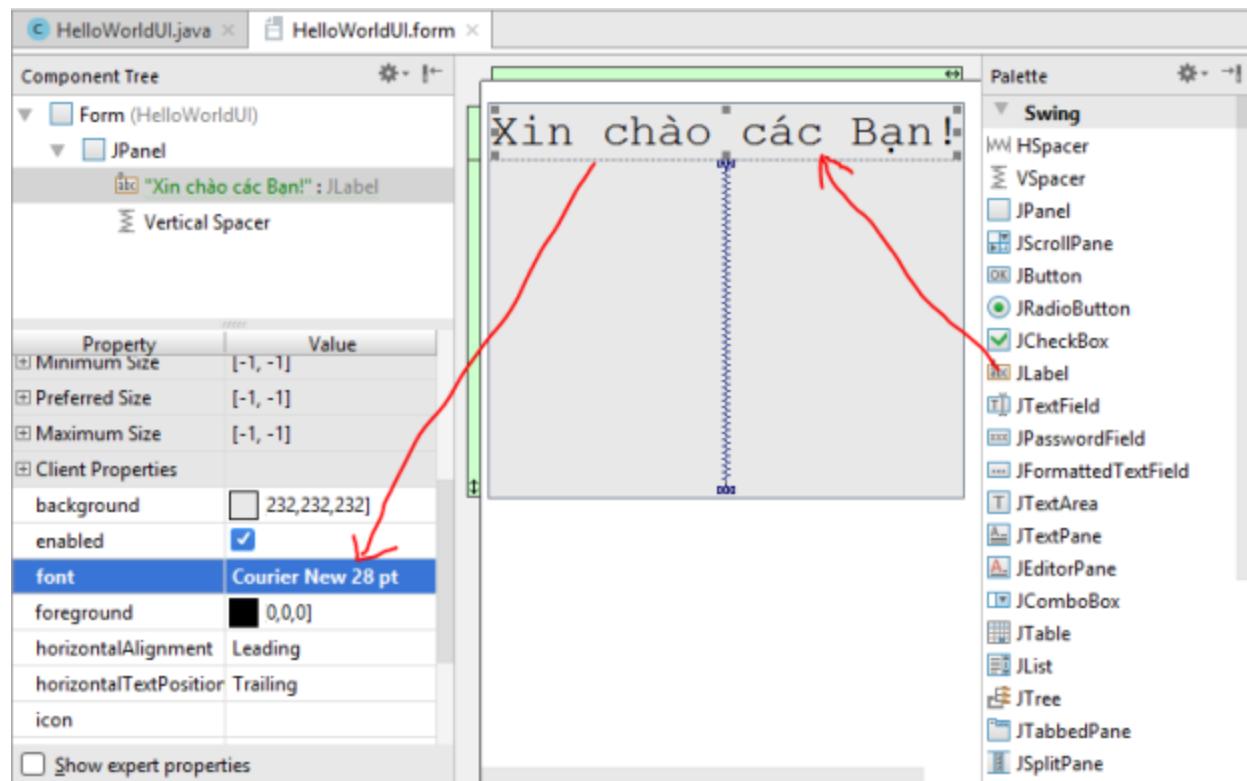
Mục 2: Cửa sổ Component Tree, là nơi cho ta biết cấu trúc layout của các control trên giao diện, nó cũng cho phép ta kéo thả control vào đây.

Mục 3: Cửa sổ Properties, là nơi cho phép ta cấu hình các thông số cho control như: Màu mè, chữ, kích thước, tên ...

Mục 4: Là nơi thiết kế giao diện, cho phép ta kéo thả control vào đây

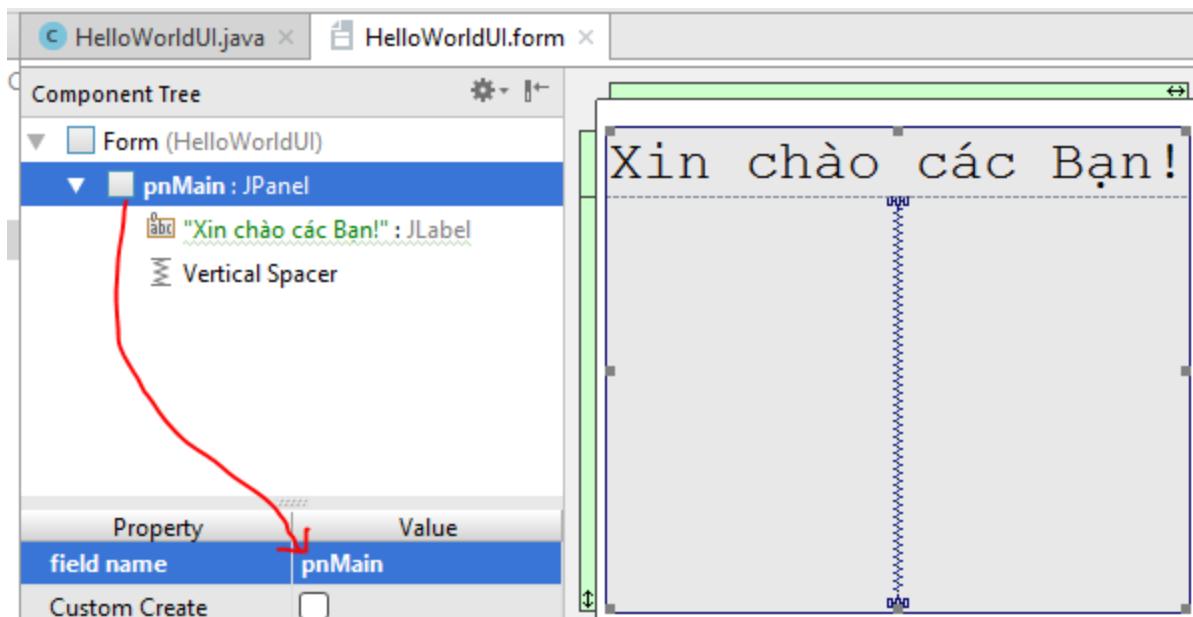
Mục 5: Cửa sổ Palette, là màn hình lưu trữ danh sách các control, nó cho phép ta Drag các Control trong này và Drop ra mục 4.

Bây giờ Ta thử kéo thả đại một vài Control ra như sau:

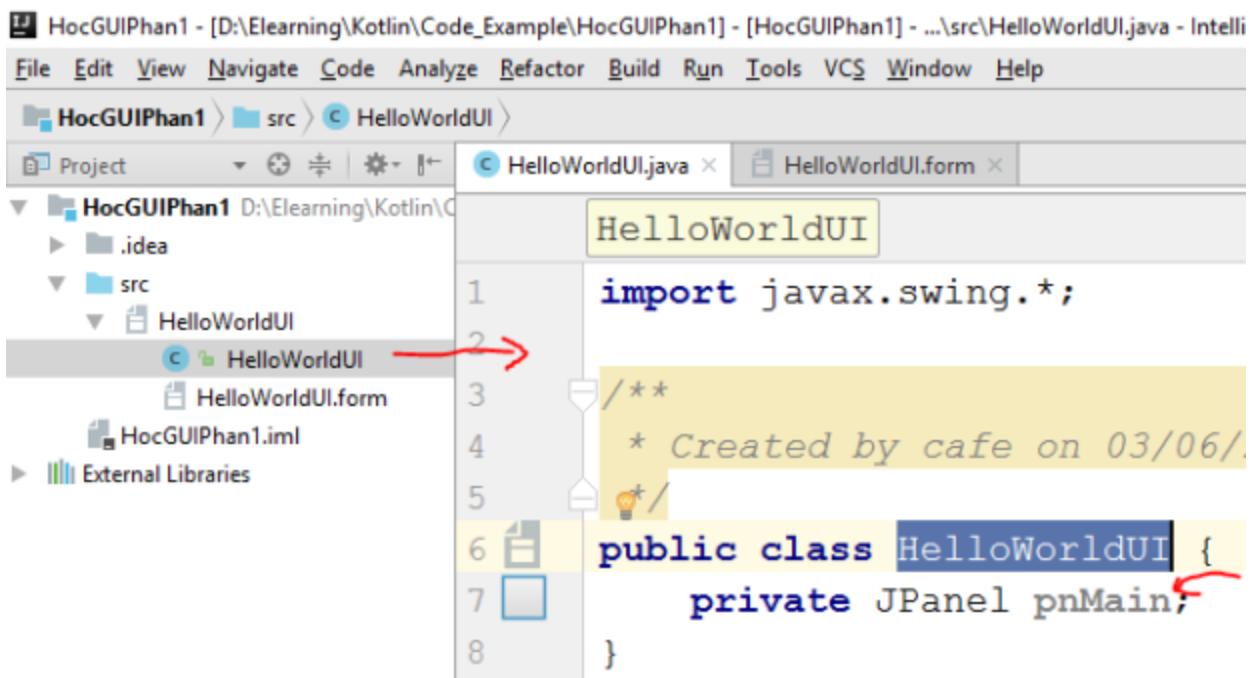


Ta túm JLabel kéo thả vào mục 4, rồi trong mục 3 chỉnh Font (bấm vào nút ...) nó sẽ ra màn hình Font ta tha hồ chọn, cấu hình sao giống như trên là OK. [Các bạn muốn hiểu chi tiết ngon ngành về Java Swing thì đăng ký học tại đây để dễ dàng học GUI cho Kotlin:](#)

Bây giờ làm sao để chạy được màn hình này? Trước tiên ta cần đặt tên cho JPanel ở ngoài cùng trước, ví dụ đặt tên thành `pnMain`:



Lúc này bạn quan sát lớp `HelloWorldUI` sẽ tự động xuất hiện khai báo biến của control này:



Sau đó sửa lại coding như sau:

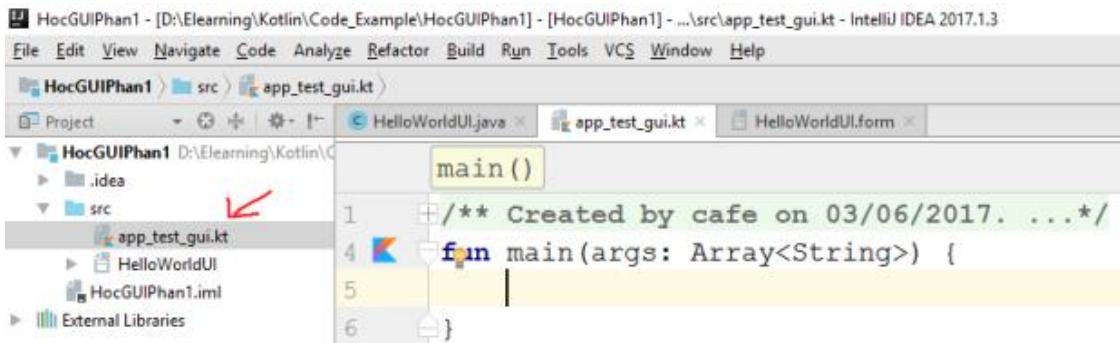
```

import javax.swing.*;

/**
 * Created by cafe on 03/06/2017.
 */
public class HelloWorldUI {
    private JPanel pnMain;
    public JPanel getPnMain()
    {
        return pnMain;
    }
}

```

Sau đó new 1 file Kotlin (app_test_gui.kt):



Bổ sung thêm các lệnh dưới đây để tải Giao diện:

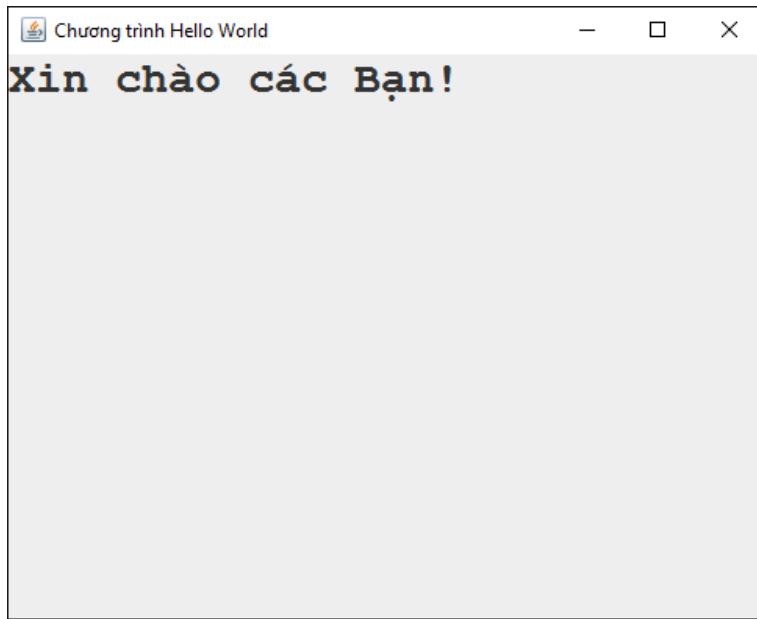
```

import javax.swing.JFrame

/**
 * Created by cafe on 03/06/2017.
 */
fun main(args: Array<String>) {
//Tạo đối tượng màn hình loại JFrame
    var helloGui:JFrame = JFrame("Chương trình Hello World")
//gán giao diện chính trong HelloWorldUI cho JFrame
    helloGui.contentPane=HelloWorldUI().pnMain
//thiết lập cho phép bấm vào X để đóng cửa sổ
    helloGui.defaultCloseOperation(JFrame.EXIT_ON_CLOSE)
//thiết lập kích thước cửa sổ
    helloGui.setSize(500,400)
//cho màn hình mặc định nằm giữa Desktop
    helloGui.setLocationRelativeTo(null)
//hiển thị giao diện:
    helloGui.isVisible=true
}

```

Chạy chương trình lên ta có giao diện:



Như vậy Tui đã hướng dẫn xong cách làm thế nào để tạo ra một giao diện trong Kotlin, bài sau Tui sẽ làm thêm 1 ví dụ về giao diện Giải phương trình bậc 1, sử dụng layout FormLayout jgoodies cũng như cách gán sự kiện cho các control. Các bạn chú ý theo dõi nhé

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/yveku2qc4vchwl/HocGUIMon1.rar>

Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 36-Thiết kế giao diện trong Kotlin – phần 2

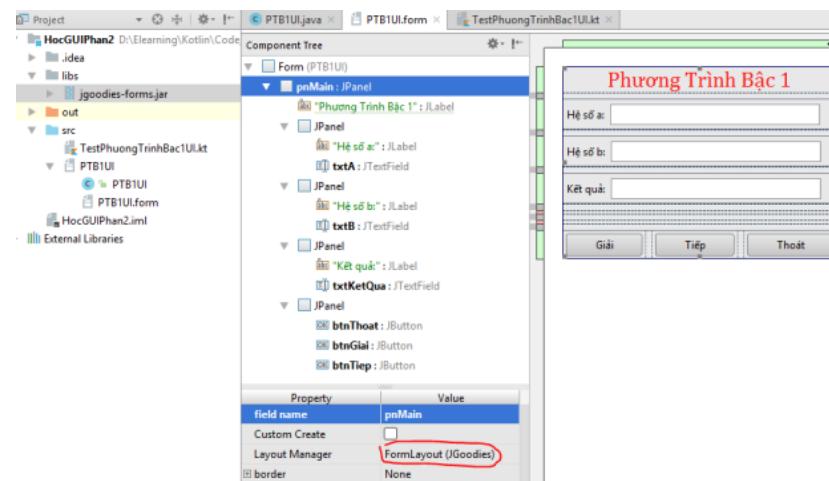
Trong [bài 35](#) Tui đã hướng dẫn các bạn cách làm thế nào để tạo được giao diện trong Kotlin. Trong bài này Tui tiếp tục làm thêm một ví dụ về thiết kế giao diện Giải Phương Trình Bậc 1, sử dụng layout FormLayout (Jgoodies) và cách gán sự kiện cho các Control. Giao diện đơn giản như sau:



Bạn vào link sau để tải thư viện:

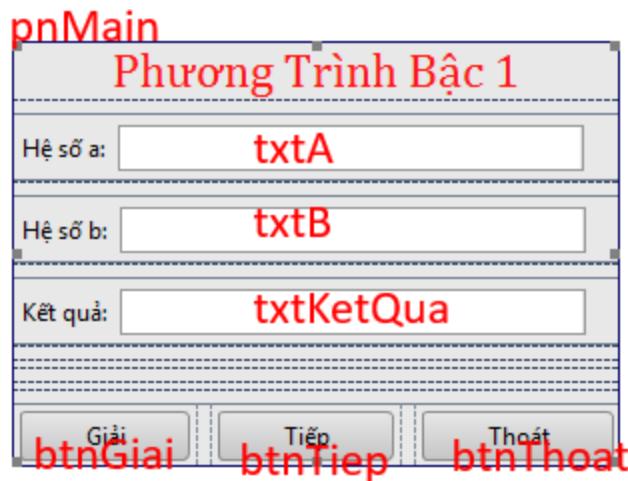
<http://www.java2s.com/Code/JarDownload/jgoodies/jgoodies-forms.jar.zip>

Giải nén ra và lấy thư viện jgoodies-forms.jar (khoảng 86kb) rồi tham chiêu nào vào Project giống như [bài 32](#) mà Tui đã hướng dẫn, sau đó thiết kế giao diện theo cấu trúc dưới đây (Ngay màn hình đầu tiên tạo Giao diện thì nhớ chọn Layout manager là FormLayout(jGoodies):



Sau đó các dòng JPanel của Hệ số a, hệ số b, kết quả, các button bạn chọn Layout Manager là FFlowLayout (tức là chỉ có pnMain mặc định là FormLayout(JGoodies)).

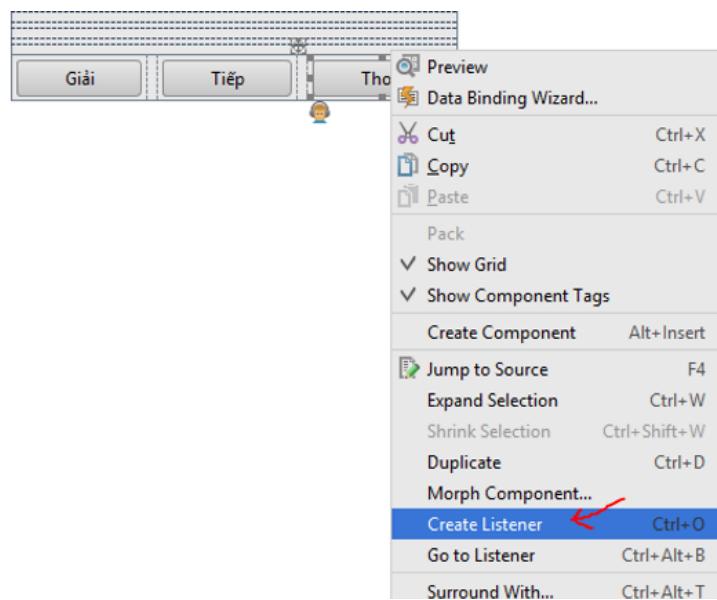
Đặt tên các Control giống như hình trên (chọn field name trong cửa sổ Property để đặt tên cho Control):



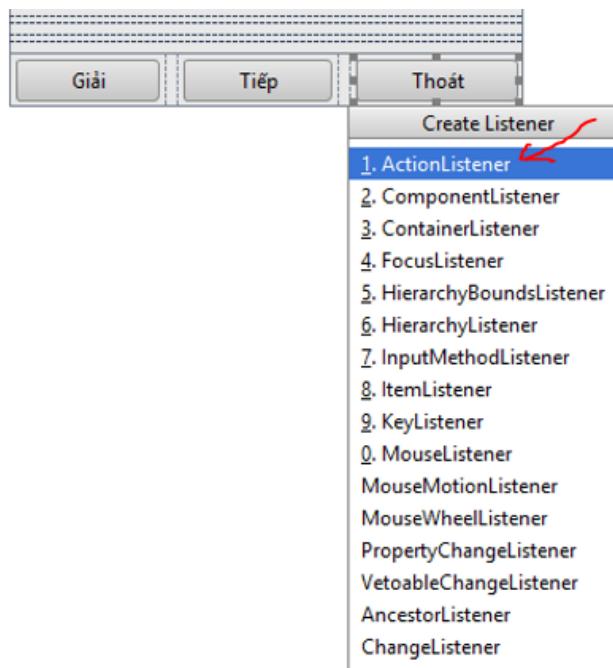
Tiếp theo gán sự kiện cho các Button:

1) Sự kiện cho Button Thoát:

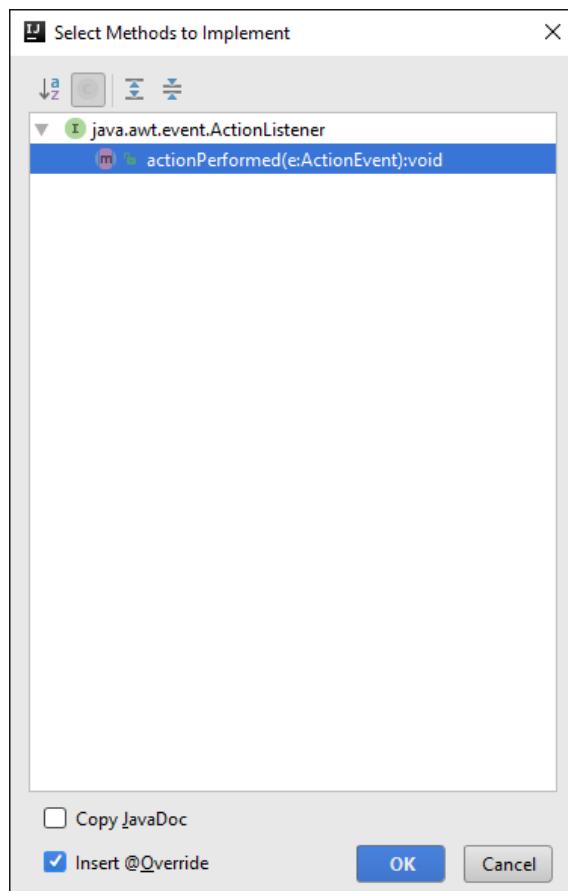
Bấm chuột phải vào Button Thoát/ chọn Create Listener:



Sau đó chọn Action Listener:



Rồi chọn actionPerformed:



Lúc này chương trình sẽ tự động xuất hiện sự kiện cho Button thoát (nó tự tạo trong Constructor):

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Created by cafe on 03/06/2017.
 */
public class PTB1UI {
    private JTextField txtA;
    private JTextField txtB;
    private JTextField txtKetQua;
    private JButton btnThoat;
    private JButton btnGiai;
    private JButton btnTiеп;
    private JPanel pnMain;

    public PTB1UI() {
        btnThoat.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }

}
```

Tương tự như vậy ta lặp gán sự kiện cho Button Giải, Button Tiếp, Đồng thời bổ sung thêm getpnMain để truy suất panel pnMain. Cuối cùng coding sẽ như sau:

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Created by cafe on 03/06/2017.
 */
public class PTB1UI {
    private JTextField txtA;
    private JTextField txtB;
    private JTextField txtKetQua;
    private JButton btnThoat;
```

```

private JButton btnGiai;
private JButton btnTiep;
private JPanel pnMain;

public PTB1UI() {
    btnThoat.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });
    btnGiai.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            double a=Double.parseDouble(txtA.getText());
            double b=Double.parseDouble(txtB.getText());
            if(a==0 && b==0)
                txtKetQua.setText("Vô số nghiệm");
            else if(a==0 && b!=0)
                txtKetQua.setText("Vô nghiệm");
            else
                txtKetQua.setText("x=" + (-b/a));
        }
    });
    btnTiep.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            txtA.setText("");
            txtB.setText("");
            txtKetQua.setText("");
            txtA.requestFocus();
        }
    });
}

public JPanel getPnMain()
{
    return pnMain;
}
}

```

Cuối cùng ta tạo một Kotlin File để chạy phần mềm (TestPhuongTrinhBac1UI.kt):

```

import javax.swing.JFrame

/*
 * Created by cafe on 03/06/2017.

```

```

    */
fun main(args: Array<String>) {
    var frm: JFrame = JFrame("Trần Duy Thành")
    frm.contentPane= PTB1UI().pnMain
    frm.defaultCloseOperation=JFrame.EXIT_ON_CLOSE
    frm.setSize(300,250)
    frm.setLocationRelativeTo(null)
    frm.isVisible=true
}

```

Chạy chương trình lên ta sẽ có giao diện như yêu cầu ở trên:



Như vậy Tui đã hướng dẫn xong cách làm thế nào để tạo ra một giao diện trong Kotlin, cách gán sự kiện, cách bố trí giao diện cũng như sử dụng Form Layout JGoodies. Bài sau Tui sẽ hướng dẫn các bạn bài JTable để hiển thị danh sách dữ liệu, Các bạn chú ý theo dõi nhé.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/066jaszvy7oo5up/HocGUIPhan2.rar>

Hẹn gặp các bạn ở những bài tiếp theo

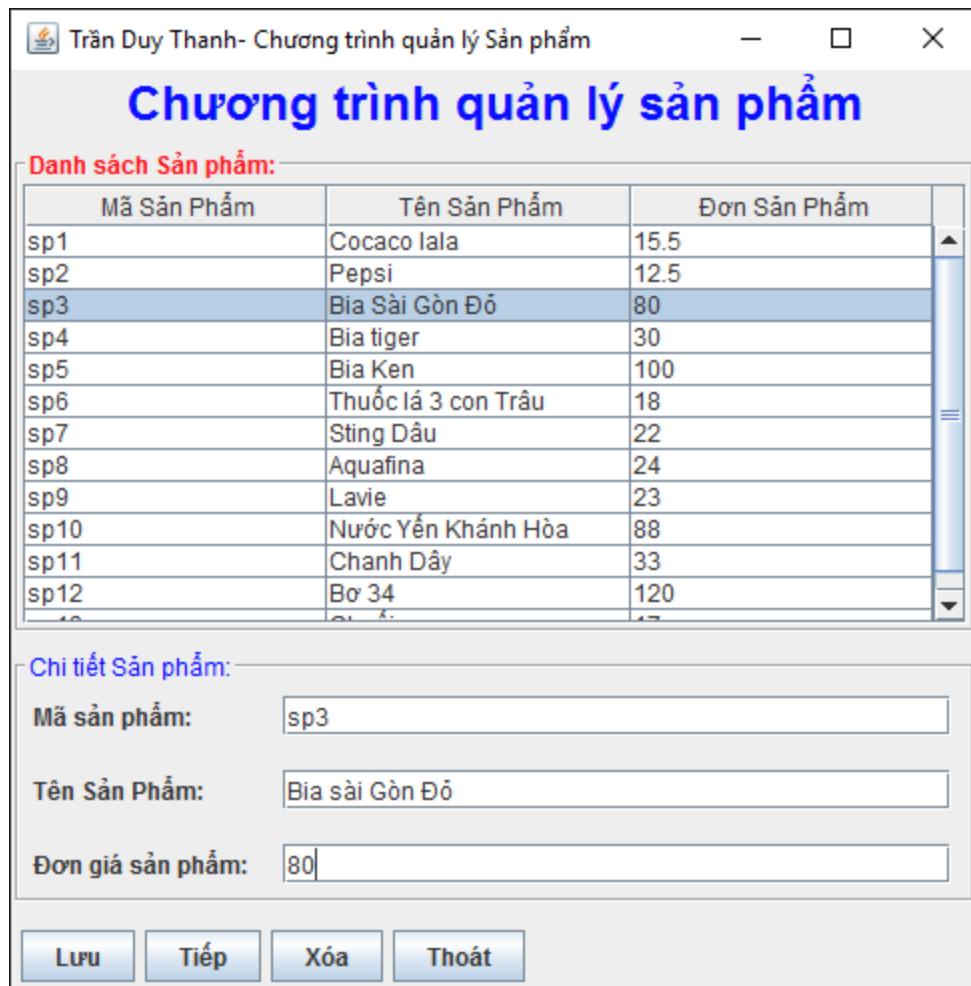
Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 37-Thiết kế giao diện trong Kotlin – phần 3

Ở [bài 35](#), [bài 36](#) Các bạn đã biết thiết kế giao diện cũng như xử lý sự kiện trong Kotlin. Trong bài này Tui tiếp tục trình bày một số control nâng cao đó là JTable để các bạn có thể hiển thị dữ liệu dạng danh sách cũng nhằm cung cấp thêm các kiến thức đã học.

Chúng ta sẽ làm bài Quản Lý sản phẩm có giao diện như dưới đây:



Bài trên sẽ có JTable để lưu trữ danh sách sản phẩm. Khi nhấp vào Sản phẩm nào thì thông tin chi tiết của nó sẽ được hiển thị vào mục chi tiết bên dưới.

Chức năng các nút:

Lưu: Đưa Sản phẩm vào JTable

Tiếp: Xóa trắng dữ liệu trong mục chi tiết và đưa con trỏ văn bản về ô nhập mã Sản phẩm

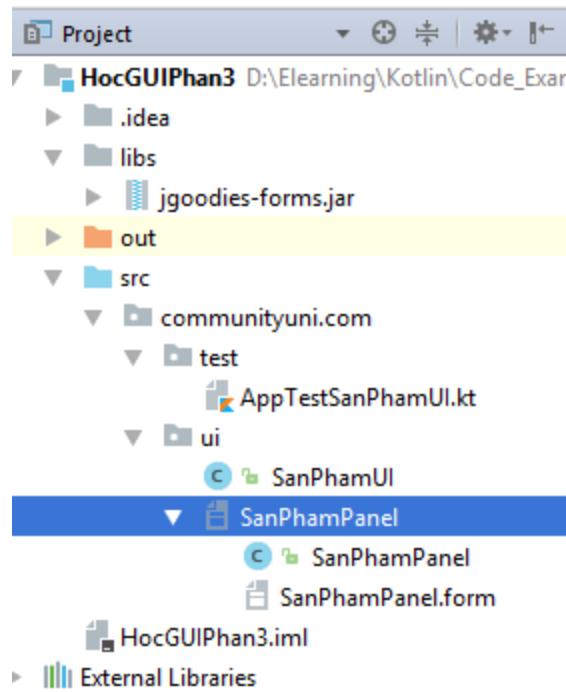
Xóa: Xóa sản phẩm nào đang chọn trên JTable

Thoát: Bấm vào để thoát phần mềm

Bài này Tui tập trung vào việc hướng dẫn cách thiết kế giao diện, gán sự kiện để các bạn thật rành về giao diện. Tui chưa tập trung vào xử lý hướng đối tượng. Bài sau Tui sẽ làm tổng hợp các kiến thức trong khóa học Kotlin từ cơ bản cho tới Lưu/đọc file trên giao diện có Menu và các Control đã học...

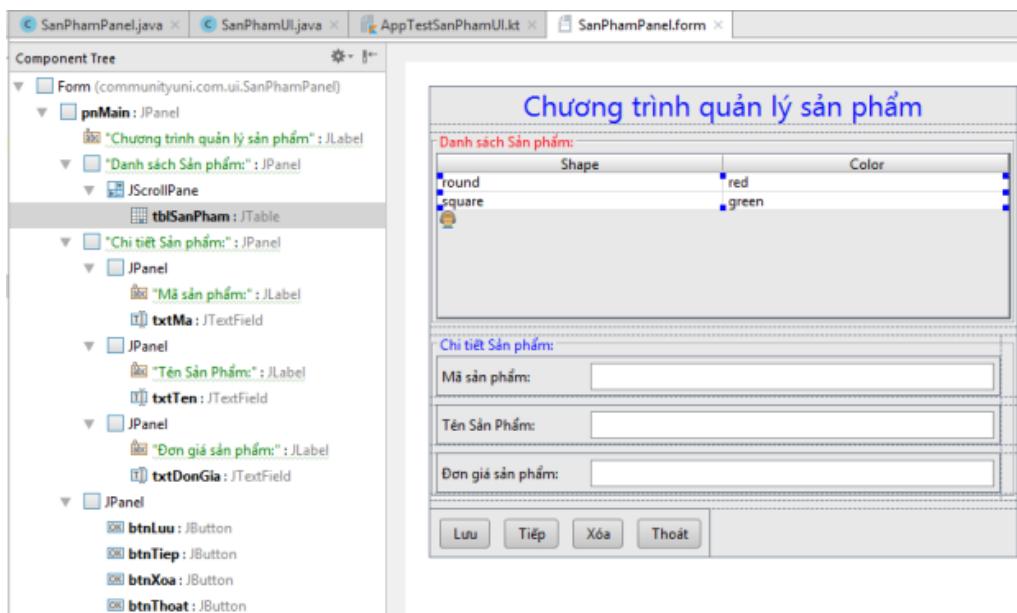
Và Tui nhắc lại là Kotlin đang dùng thư viện JVM, nên các bạn muốn thực sự hiểu sâu về Java để hỗ trợ cho Kotlin thì các bạn có thể [đăng ký học khóa Lập trình Java nâng cao tại đây](#) hoặc [tại đây](#). Kotlin có thể gọi Java và Java có thể gọi được Kotlin, Android cũng dùng cả 2 ngôn ngữ này để lập trình nên các bạn cần biết cả 2.

Ta tạo 1 Project tên là HocGUIPhan3 có, đưa thư việt Jgoodies, tạo package như hình dưới đây:



Các giao diện sẽ nằm trong package: communityuni.com.ui . Chú ý là giao diện SanPhamPanel khi bạn tạo ra thì nó tự động chia làm 2 Lớp (Lớp SanPhamPanel.form là nơi chúng ta thiết kế kéo thả giao diện, lớp SanPhamPanel là nơi cho ta xử lý các nghiệp vụ cũng như khởi tạo thêm các custom component)

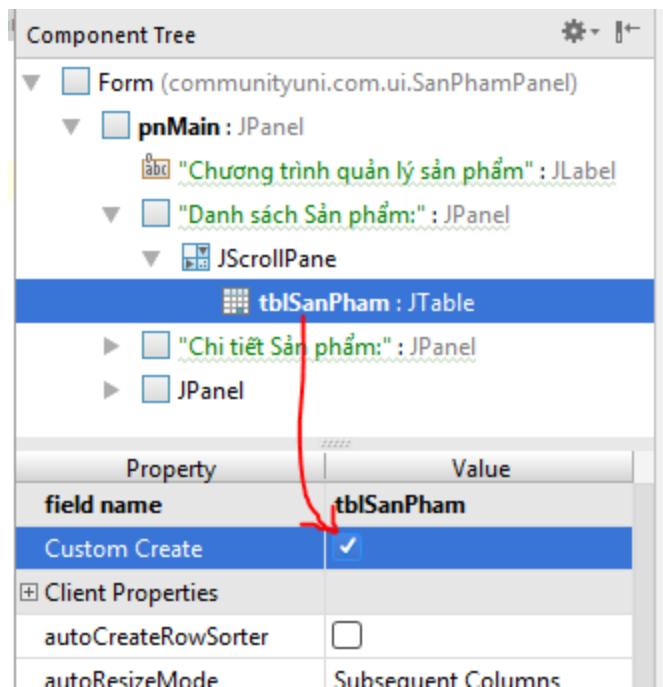
Tập tin chứa hàm main để chạy phần mềm nằm trong package: communityuni.com.test trong package communityuni.com.ui bạn tạo một Form có tên SanPhamPanel để thiết kế giao diện như dưới đây (khoan hãy để ý tới lớp SanPhamUI, Tui sẽ trình bày sau):



Bạn nhìn vào màn hình Component Tree để biết cách bố trí các Control trên giao diện, cũng như đặt tên chính xác cho chúng.

Chú ý là JPanel pnMain bạn chọn layout manager là FormLayout Jgoodies, các JPanel còn lại chọn Layout manager là **FlowLayout** hết nhé các bạn.

Bây giờ bạn chọn Jtable (tblSanPham) rồi checked vào Custom Create:



Lúc này chương trình sẽ tự động phát sinh ra hàm `createUIComponents()` trong lớp xử lý nghiệp vụ:

```

HocGUIPhan3 - [D:\Elearning\Kotlin\Code_Example\HocGUIPhan3] - [HocGUIPhan3] - ...\\src\\communityuni\\com\\ui\\SanPhamPanel.java - IntelliJ IDEA 2017.1
Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HocGUIPhan3 > src > communityuni > com > ui > SanPhamPanel
project > HocGUIPhan3 D:\Elearning\Kotlin\Code_Exa
  .idea
  libs
  > jgoodies-forms.jar
  out
  src
    communityuni.com
      test
        AppTestSanPhamUI.kt
      ui
        SanPhamUI
        SanPhamPanel
          SanPhamPanel
          SanPhamPanel.form
  HocGUIPhan3.iml
External Libraries

double gia=Double.parseDouble(txtDonGia.getText());
txtMa.setText(ma);
txtTen.setText(ten);
txtDonGia.setText(gia+"");
}

private void createUIComponents() {
}

public JPanel getPnMain()
{
    return pnMain;
}
}

```

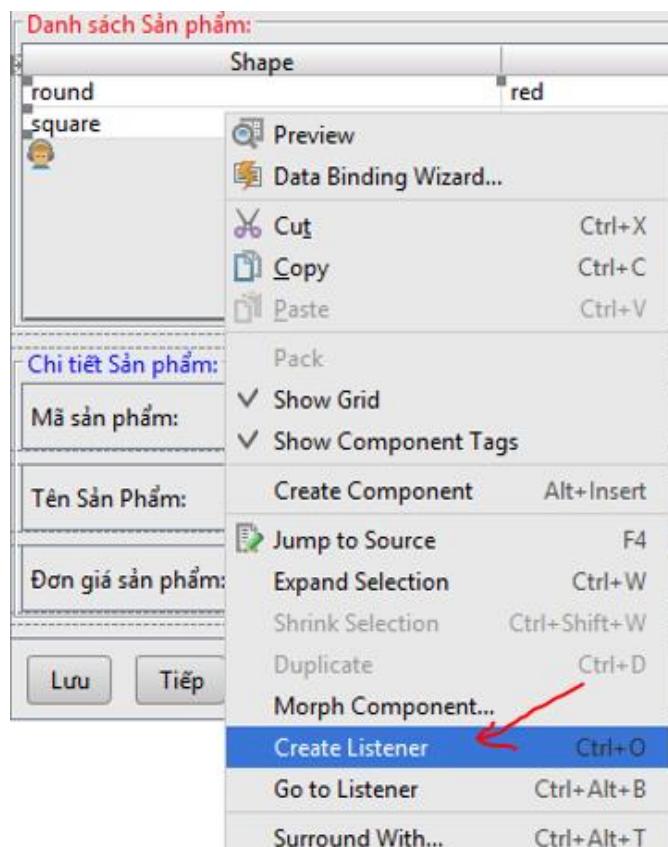
Ta sẽ khởi tạo và hiệu chỉnh giao diện bằng coding trong hàm này, Control nào muốn được thay đổi bằng coding thì cứ checked vào Custom Create.

Có rất nhiều cách tạo bảng, ở đây ta dùng `DefaultTableModel` , để tạo bảng ta có thể chỉnh như sau:

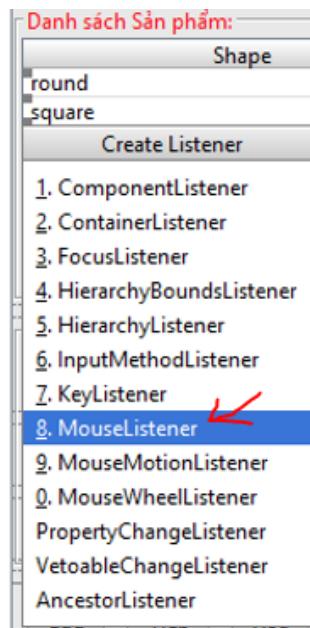
```
private JTable tblSanPham;
private DefaultTableModel tableModelSanPham;
private void createUIComponents() {
    tableModelSanPham=new DefaultTableModel();
    tableModelSanPham.addColumn("Mã Sản Phẩm");
    tableModelSanPham.addColumn("Tên Sản Phẩm");
    tableModelSanPham.addColumn("Đơn giá Sản Phẩm");
    tblSanPham=new JTable(tableModelSanPham);
}
```

Ở trên là tạo 1 `JTable` có 3 cột (mã, tên và đơn giá).

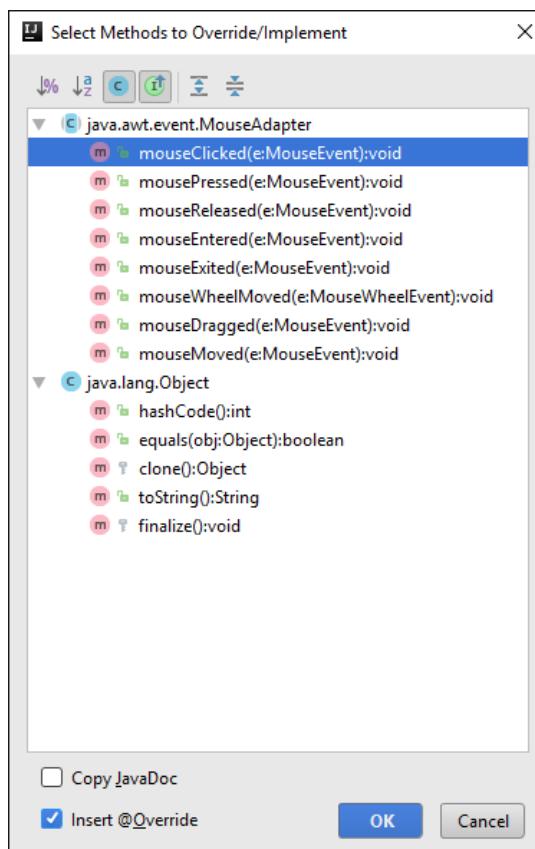
Để gán sự kiện cho `Jtable` ta làm tương tự như gán sự kiện cho `JButton` mà Tui đã hướng dẫn ở [bài 36](#) (do đó Tui không có nói lại cách gán sự kiện của các `JButton`, các bạn tự xử): Ta bấm chuột phải vào `Jtable` rồi chọn Create Listener:



Sau đó chọn Mouse listener:



Sau đó chọn Mouse pressed hoặc mouse click:



Bấm Ok để tạo, lúc này ta có sự kiện Mouse click:

```

tblSanPham.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        super.mousePressed(e);

    }
}) ;

```

Ta bổ sung coding cho sự kiện này để lấy hiển thị thông tin chi tiết lên các JTextField:

```

tblSanPham.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        super.mousePressed(e);
        int row= tblSanPham.getSelectedRow();
        String ma=tblSanPham.getValueAt(row,0).toString();
        String ten=tblSanPham.getValueAt(row,1).toString();
        double
gia=Double.parseDouble(tblSanPham.getValueAt(row,2).toString());
        txtMa.setText(ma);
        txtTen.setText(ten);
        txtDonGia.setText(gia+"");
    }
}) ;

```

Cuối cùng ta bổ sung cho toàn bộ các sự kiện của các JButton trên giao diện trong lớp xử lý nghiệp vụ như sau:

```

package communityuni.com.ui;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Vector;

/**
 * Created by cafe on 04/06/2017.
 */
public class SanPhamPanel {
    private JPanel pnMain;
    private JTable tblSanPham;
    private DefaultTableModel tableModelSanPham;
    private JButton btnLuu;

```

```

private JButton btnTiep;
private JButton btnXoa;
private JButton btnThoat;
private JTextField txtMa;
private JTextField txtTen;
private JTextField txtDonGia;

public SanPhamPanel() {
    btnTiep.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            txtMa.setText("");
            txtTen.setText("");
            txtDonGia.setText("");
            txtMa.requestFocus();
        }
    });
    btnThoat.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });
    btnLuu.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Vector<String>vector=new Vector<>();
            vector.add(txtMa.getText());
            vector.add(txtTen.getText());
            vector.add(txtDonGia.getText());
            tableModelSanPham.addRow(vector);
        }
    });
    tblSanPham.addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            super.mousePressed(e);
            int row=tblSanPham.getSelectedRow();
            String
ma=tblSanPham.getValueAt(row,0).toString();
            String
ten=tblSanPham.getValueAt(row,1).toString();
            double
gia=Double.parseDouble(tblSanPham.getValueAt(row,2).toString());
            txtMa.setText(ma);
            txtTen.setText(ten);
            txtDonGia.setText(gia+"");
        }
    });
}

```

```

        }
    });
tblSanPham.addMouseListener(new MouseAdapter() {
});
btnXoa.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (tblSanPham.getSelectedRow() >= 0)

tableModelSanPham.removeRow(tblSanPham.getSelectedRow());
    }
});
}

private void createUIComponents() {
    tableModelSanPham=new DefaultTableModel();
    tableModelSanPham.addColumn("Mã Sản Phẩm");
    tableModelSanPham.addColumn("Tên Sản Phẩm");
    tableModelSanPham.addColumn("Đơn Giá Sản Phẩm");
    tblSanPham=new JTable(tableModelSanPham);
}
public JPanel getPnMain()
{
    return pnMain;
}
}
}

```

Bây giờ trong package communityuni.com.ui ta tạo một lớp giao diện tên SanPhamUI kế thừa JFrame có coding như sau:

```

package communityuni.com.ui;

import javax.swing.*;

< /**
 * Created by cafe on 04/06/2017.
 */

public class SanPhamUI extends JFrame {
    public SanPhamUI(String title)
    {
        super(title);
        setContentPane(new SanPhamPanel().getPnMain());
    }
    public void showWindow()
    {
        setSize(500,500);
    }
}

```

```

        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setVisible(true);
    }
}

```

Cuối cùng trong package communityuni.com.test ta tạo 1 file Kotlin AppTestSanPhamUI.kt để chạy phần mềm:

```

package communityuni.com.test

import communityuni.com.ui.SanPhamUI

/*
 * Created by cafe on 04/06/2017.
 */
fun main(args: Array<String>) {
    var gui:SanPhamUI= SanPhamUI("Trần Duy Thành- Chương trình
quản lý Sản phẩm")
    gui.showWindow()
}

```

Chạy chương trình lên ta sẽ có giao diện như Tui đã yêu cầu ở trên.

Như vậy Tui đã hướng dẫn xong cách làm thế nào để tạo ra một giao diện có JTable trong Kotlin, cách gán sự kiện cho JTable, cách bố trí giao diện cũng như sử dụng Form Layout JGoodies, Flow Layout. Bài sau Tui sẽ hướng dẫn các bạn bài tổng hợp Kotlin từ cơ bản tới Hướng đối tượng, lưu file, đọc file, Menu trong Kotlin để hiển thị và lưu trữ danh sách dữ liệu, Các bạn chú ý theo dõi nhé.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/ko320gej2mkl737/HocGUIMain.rar>

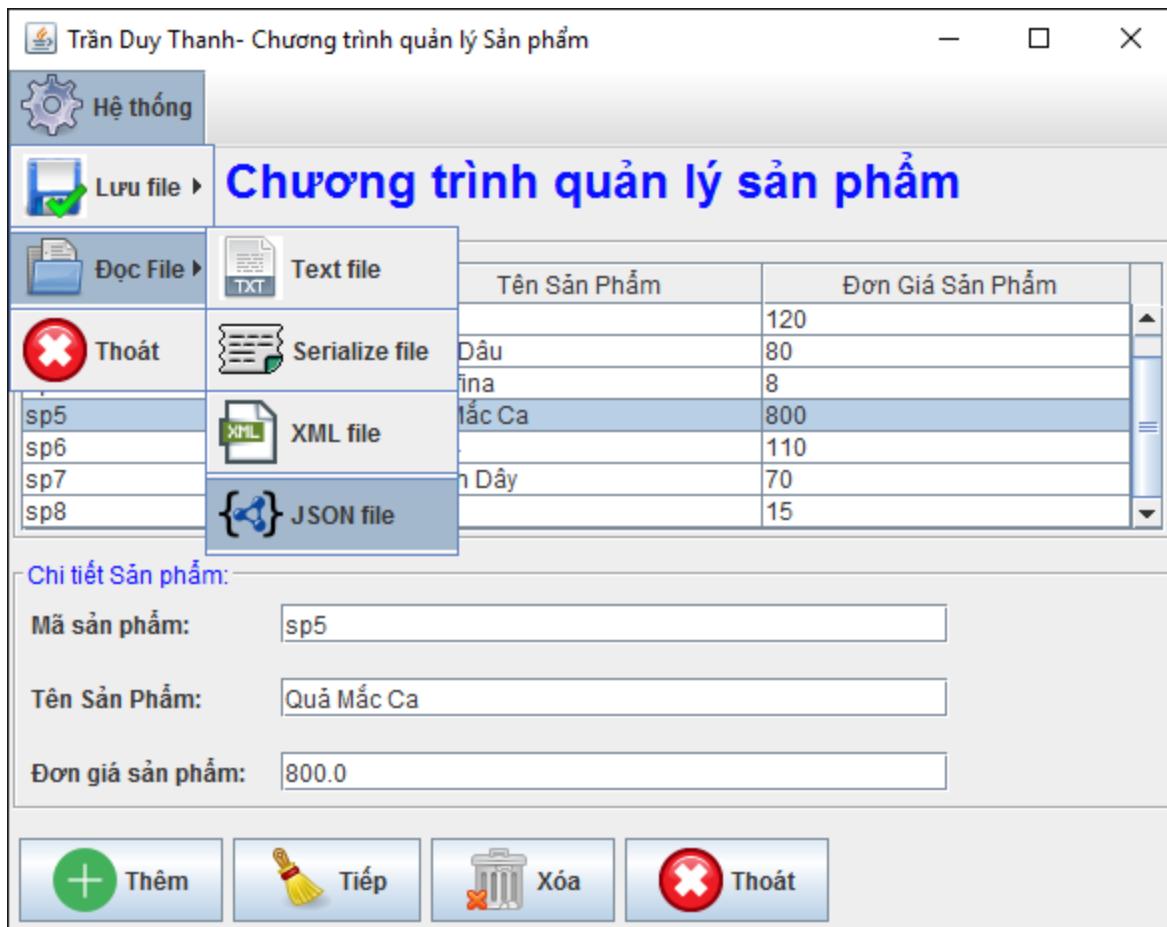
Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 38-Thiết kế giao diện trong Kotlin – phần 4

Ở [Bài 37-Thiết kế giao diện trong Kotlin – phần 3](#) bạn đã xây dựng được một phần mềm quản lý Sản phẩm nhưng chưa liên quan tới OOP, tương tác File , JMenu và JFileChooser. Trong bài này Tui sẽ tiếp tục phát triển [bài 37](#), nên bạn nào chưa học [bài 37](#) thì tự vào học để lấy Code tiếp tục phát triển nhé, giao diện sẽ bổ sung thêm JMenu để lưu file và mở File:

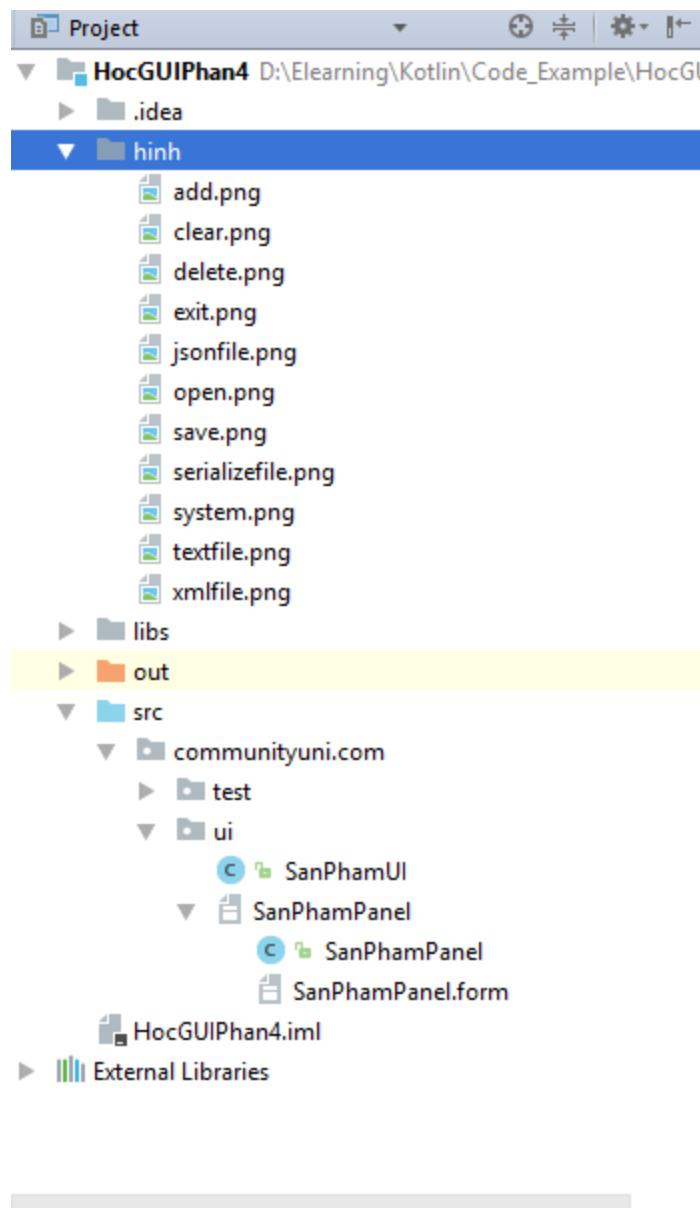


Bài này Tui bổ sung thêm các Menu Lưu File và Mở file với 4 loại mà chúng ta đã được học trước đó ([Text File](#), [Serialize File](#), [XML file](#), [JSON File](#)) cũng như các Icon cho các Button.

Chương trình sẽ dùng JFileChooser để cho phép người dùng chỉ định nơi lưu trữ, đặc biệt phải dùng lập trình hướng đối tượng để xử lý. Bài học này khá khó nên các bạn phải chú ý làm theo hướng dẫn và chịu khó suy luận.

Bây giờ Tui hướng dẫn chi tiết cách thực hiện:

Trước tiên bạn lôi cái Project làm ở [bài 37](#) ra, bổ sung thêm danh sách các hình ảnh sau(bạn tạo một thư mục **hình**, các hình bạn tự tải rồi lưu vào đây, lấy đại cho nó lành):



Bổ sung IIcon cho các Button trong SanPhamPanel bằng cách tạo một hàm createIIcon() có coding như sau(Tui lược bỏ các coding cũ, chút nữa sẽ có coding đầy đủ):

```

public SanPhamPanel() {
    //Tui hide coding .... cho dẽ nhìn
    createIcon();
}

private void createIcon() {
    btnLuu.setIcon(new ImageIcon("hinh/add.png"));
    btnTiep.setIcon(new ImageIcon("hinh/clear.png"));
    btnXoa.setIcon(new ImageIcon("hinh/delete.png"));
    btnThoat.setIcon(new ImageIcon("hinh/exit.png"));
}

```

Tiếp tục vào [SanPhamUI](#) để bổ sung các Menu và Icon cho nó (Tui lược bỏ các coding cũ, chút nữa sẽ có coding đầy đủ):

```

public class SanPhamUI extends JFrame {
    JMenuBar mnuBar;
    JMenu mnuSystem;
    JMenu mnuSystemSave;
    JMenuItem mnuSystemSaveTextFile;
    JMenuItem mnuSystemSaveSerializeFile;
    JMenuItem mnuSystemSaveXMLFile;
    JMenuItem mnuSystemSaveJSONFile;
    JMenu mnuSystemOpen;
    JMenuItem mnuSystemOpenTextFile;
    JMenuItem mnuSystemOpenSerializeFile;
    JMenuItem mnuSystemOpenXMLFile;
    JMenuItem mnuSystemOpenJSONFile;
    JMenuItem mnuSystemExit;
    public SanPhamUI(String title)
    {
        super(title);
        setContentPane(new SanPhamPanel().getPnMain());
        createMenu();
        createIcon();
    }
    public void createMenu()
    {
        mnuBar=new JMenuBar();
        setJMenuBar(mnuBar);
        mnuSystem=new JMenu("Hệ thống");
        mnuBar.add(mnuSystem);

        mnuSystemSave=new JMenu("Lưu file");
        mnuSystem.add(mnuSystemSave);
        mnuSystemSaveTextFile=new JMenuItem("Text file");

```

```

        mnuSystemSave.add(mnuSystemSaveTextFile);
        mnuSystemSave.addSeparator();
        mnuSystemSaveSerializeFile=new JMenuItem("Serialize
file");
        mnuSystemSave.add(mnuSystemSaveSerializeFile);
        mnuSystemSave.addSeparator();
        mnuSystemSaveXMLFile=new JMenuItem("XML file");
        mnuSystemSave.add(mnuSystemSaveXMLFile);
        mnuSystemSave.addSeparator();
        mnuSystemSaveJSONFile=new JMenuItem("JSON file");
        mnuSystemSave.add(mnuSystemSaveJSONFile);

        mnuSystem.addSeparator();

        mnuSystemOpen=new JMenu("Đọc File");
        mnuSystem.add(mnuSystemOpen);
        mnuSystemOpenTextFile=new JMenuItem("Text file");
        mnuSystemOpen.add(mnuSystemOpenTextFile);
        mnuSystemOpen.addSeparator();
        mnuSystemOpenSerializeFile=new JMenuItem("Serialize
file");
        mnuSystemOpen.add(mnuSystemOpenSerializeFile);
        mnuSystemOpen.addSeparator();
        mnuSystemOpenXMLFile=new JMenuItem("XML file");
        mnuSystemOpen.add(mnuSystemOpenXMLFile);
        mnuSystemOpen.addSeparator();
        mnuSystemOpenJSONFile=new JMenuItem("JSON file");
        mnuSystemOpen.add(mnuSystemOpenJSONFile);

        mnuSystem.addSeparator();

        mnuSystemExit=new JMenuItem("Thoát");
        mnuSystem.add(mnuSystemExit);
    }

    private void createIcon() {
        mnuSystem.setIcon(new ImageIcon("hinh/system.png"));
        mnuSystemSave.setIcon(new ImageIcon("hinh/save.png"));
        mnuSystemSaveTextFile.setIcon(new
ImageIcon("hinh/textfile.png"));
        mnuSystemSaveSerializeFile.setIcon(new
ImageIcon("hinh/serializefile.png"));
        mnuSystemSaveXMLFile.setIcon(new
ImageIcon("hinh/xmlfile.png"));
        mnuSystemSaveJSONFile.setIcon(new
ImageIcon("hinh/jsonfile.png"));

        mnuSystemOpen.setIcon(new ImageIcon("hinh/open.png"));
    }
}

```

```

        mnuSystemOpenTextFile.setIcon(new
ImageIcon ("hinh/textfile.png"));
        mnuSystemOpenSerializeFile.setIcon(new
ImageIcon ("hinh/serializefile.png"));
        mnuSystemOpenXMLFile.setIcon(new
ImageIcon ("hinh/xmlfile.png"));
        mnuSystemOpenJSONFile.setIcon(new
ImageIcon ("hinh/jsonfile.png"));

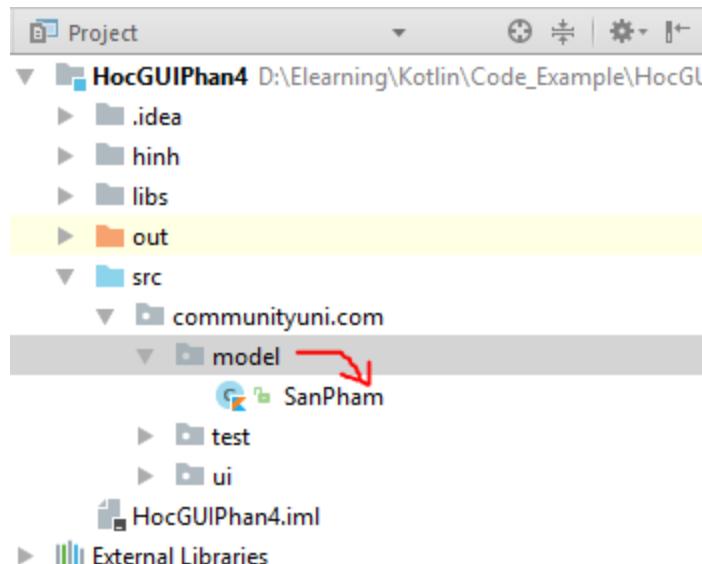
        mnuSystemExit.setIcon(new ImageIcon ("hinh/exit.png"));
    }
}

```

Khi bạn bổ sung xong các coding ở trên thì sẽ có được giao diện như Tui yêu cầu ở trên.

Bây giờ Ta xử lý Coding hướng đối tượng Kotlin cho Sản Phẩm (tạo class SanPham), và 4 loại tập tin (Tui sẽ viết mô hình lớp kế thừa dẫn xuất từ Interface).

Trước tiên bạn tạo một package tên là communityuni.com.model, có chứa Lớp Sản phẩm với cấu trúc như sau:



```

package communityuni.com.model

import java.io.Serializable

/**
 * Created by cafe on 04/06/2017.
 */

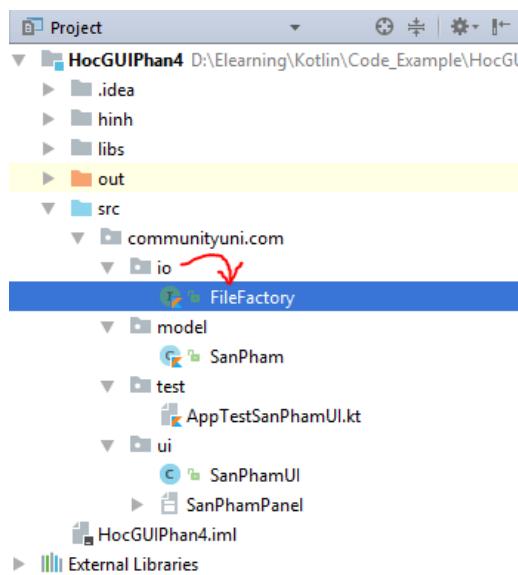
```

```

class SanPham:Serializable {
    private var ma:String="";
    private var ten:String="";
    private var gia:Double=0.0;
    constructor()
    constructor(ma: String, ten: String, gia: Double) {
        this.ma = ma
        this.ten = ten
        this.gia = gia
    }
    public var Ma:String
        get() = ma
        set(value) {ma=value}
    public var Ten:String
        get() = ten
        set(value) {ten=value}
    public var Gia:Double
        get() = gia
        set(value) {gia=value}
    override fun toString(): String {
        return "$ma\t$ten\t$gia"
    }
}

```

Để xử lý lưu và đọc tập tin ta tạo package communityuni.com.io, tạo một interface FileFactory bằng Kotlin như hình dưới đây:



```

package communityuni.com.io

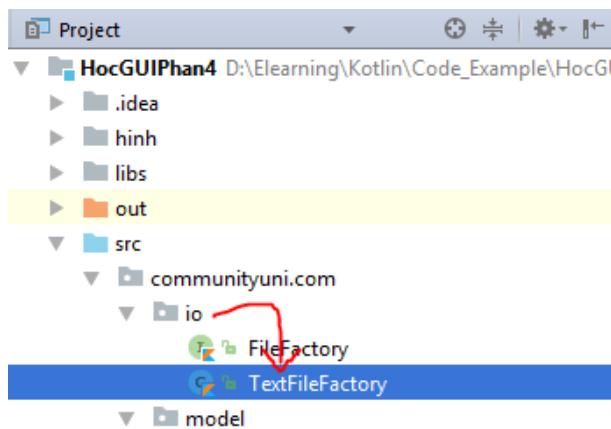
import communityuni.com.model.SanPham

/**
 * Created by cafe on 04/06/2017.
 */
interface FileFactory {
    /**
     * @author Trần Duy Thành
     * @param data: Dữ liệu là Danh sách sản phẩm muốn lưu
     * @param path: Đường dẫn lưu trữ
     * @return true nếu lưu thành công, false nếu lưu thất bại
     */
    fun LuuFile(database:MutableList<SanPham>, path:String) :Boolean
    /**
     * @author Trần Duy Thành
     * @param path:đường dẫn muốn đọc dữ liệu
     * @return Danh sách sản phẩm MutableList
     */
    fun DocFile(path:String) :MutableList<SanPham>
}

```

Sau Đó bạn tạo 4 Lớp để xử lý 4 loại tập tin, các lớp này bạn đã được học ở các bài:
[Text File](#), [Serialize File](#), [XML file](#),[JSON File](#).

Lớp TextFileFactory dẫn xuất từ Interface FileFactory ở trên để lưu và đọc TextFile:



Chú ý khi bạn cho dẫn xuất từ Interface thì như lý thuyết đã học, bạn phải Override toàn bộ phương thức trừu tượng trong Interface nhé (chỉ cần nhấn tổ hợp phím Alt+ Enter nó sẽ tự động làm giúp ta):

```

package communityuni.com.io

import communityuni.com.model.SanPham
import java.io.*

/**
 * Created by cafe on 04/06/2017.
 */
class TextFileFactory: FileFactory {
    override fun LuuFile(database: MutableList<SanPham>, path: String): Boolean {
        try {
            val fos = FileOutputStream(path)
            val osw = OutputStreamWriter(fos, "UTF-8")
            val bw = BufferedWriter(osw)
            for (sp in database) {
                bw.write(sp.toString());
                bw.newLine();
            }
            bw.close();
            osw.close();
            fos.close();
            return true
        }
        catch (ex:Exception)
        {
            ex.printStackTrace()
        }
        return false
    }

    override fun DocFile(path: String): MutableList<SanPham> {
        var data:MutableList<SanPham> = mutableListOf()
        try {
            val fis = FileInputStream(path)
            val isr = InputStreamReader(fis, "UTF-8")
            val br = BufferedReader(isr)

            var line = br.readLine()
            while (line != null) {
                var arr = line.split("\t")
                if (arr.size == 3) {
                    var sp: SanPham = SanPham()
                    sp.Ma = arr[0]
                    sp.Ten = arr[1]
                    sp.Gia = arr[2].toDouble()
                    data.add(sp)
                }
            }
        }
        catch (ex:Exception)
        {
            ex.printStackTrace()
        }
        return data
    }
}

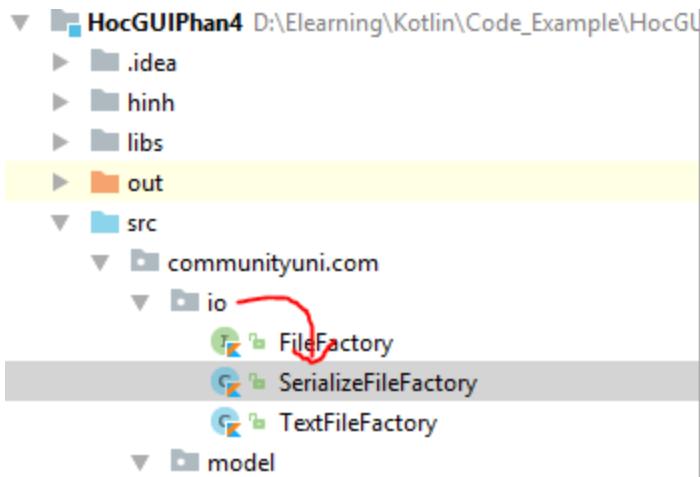
```

```

        }
        line = br.readLine()
    }
    br.close()
    isr.close()
    fis.close()
}
catch (ex:Exception)
{
    ex.printStackTrace()
}
return data
}
}

```

Tương tự cho Serialize File:



```

package communityuni.com.io
import java.io.FileInputStream
import java.io.FileOutputStream
import java.io.ObjectInputStream
import java.io.ObjectOutputStream
import communityuni.com.model.SanPham

/**
 * Created by cafe on 04/06/2017.
 */
class SerializeFileFactory : FileFactory{
    override fun LuuFile(database: MutableList<SanPham>, path:
String): Boolean {
        try {
            var fos=FileOutputStream(path);

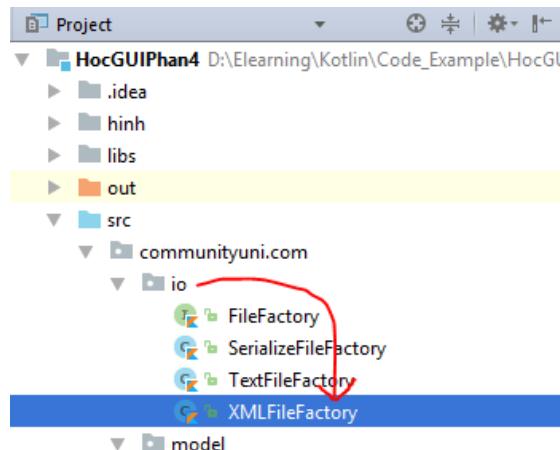
```

```

        var oos=ObjectOutputStream(fos);
        oos.writeObject(database);
        oos.close();
        fos.close();
        return true
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return false
}
override fun DocFile(path: String): MutableList<SanPham> {
    var database:MutableList<SanPham> = mutableListOf()
    try
    {
        var fis=FileInputStream(path);
        var ois=ObjectInputStream(fis);
        var obj=ois.readObject();
        database= obj as MutableList<SanPham>;
        ois.close();
        fis.close();
    }
    catch (ex:Exception)
    {
        ex.printStackTrace()
    }
    return database
}
}

```

Tương tự với lớp Kotlin XML File:



```

package communityuni.com.io

import communityuni.com.model.SanPham
import java.io.File
import javax.xml.parsers.DocumentBuilderFactory
import javax.xml.transform.stream.StreamResult
import javax.xml.transform.dom.DOMSource
import javax.xml.transform.TransformerFactory
import org.w3c.dom.Element
/**
 * Created by cafe on 04/06/2017.
 */
class XMLFileFactory:FileFactory {
    override fun LuuFile(database: MutableList<SanPham>, path:
String): Boolean {
        try {
            val docFactory =
DocumentBuilderFactory.newInstance()
            val docBuilder = docFactory.newDocumentBuilder()
// root elements
            val doc = docBuilder.newDocument()
            val rootElement = doc.createElement("SanPhams")
            doc.appendChild(rootElement)
            for(sp in database) {
                val sanPhamElement =
doc.createElement("SanPham")
                val maElement=doc.createElement("Ma")
                maElement.textContent=sp.Ma
                sanPhamElement.appendChild(maElement)
                val tenElement=doc.createElement("Ten")
                tenElement.textContent=sp.Ten
                sanPhamElement.appendChild(tenElement)
                val giaElement=doc.createElement("Gia")
                giaElement.textContent=sp.Gia.toString()
                sanPhamElement.appendChild(giaElement)
                rootElement.appendChild(sanPhamElement);
            }
// write the content into xml file
            val transformerFactory =
TransformerFactory.newInstance()
            val transformer =
transformerFactory.newTransformer()
            val source = DOMSource(doc)

            val result = StreamResult(File(path).absolutePath)
        }
    }
}

```

```

// Output to console for testing
// StreamResult result = new StreamResult(System.out);
    transformer.transform(source, result)
    return true
}
catch (ex:Exception)
{
    ex.printStackTrace()
}
return false
}

override fun DocFile(path: String): MutableList<SanPham> {
    var database:MutableList<SanPham> = mutableListOf()
    try {
//Get the DOM Builder Factory
        val factory = DocumentBuilderFactory.newInstance()

//Get the DOM Builder
        val builder = factory.newDocumentBuilder()

//Load and Parse the XML document
//document contains the complete XML as a Tree.
        val xmlfile = File(path)

        val document = builder.parse(xmlfile)

//Iterating through the nodes and extracting the data.
        val nodeList = document.documentElement.childNodes

        for (i in 0..nodeList.length - 1) {

//We have encountered an <SanPham> tag.
            val node = nodeList.item(i)
            if (node is Element) {
                val sp = SanPham()
                val childNodes = node.getChildNodes()
                for (j in 0..childNodes.getLength() - 1) {
                    val cNode = childNodes.item(j)

//Identifying the child tag of employee encountered.
                    if (cNode is Element) {
                        val content =
cNode.getLastChild().gettextContent().trim()
                        when (cNode.getNodeName()) {
                            "Ma" -> sp.Ma= content

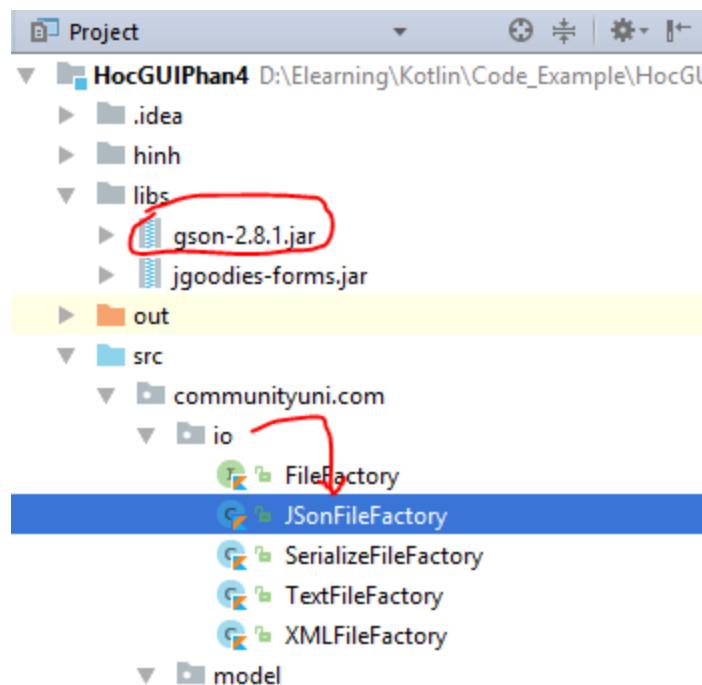
```

```

        "Ten" -> sp.Ten= content
        "Gia" -> sp.Gia=
content.toDouble()
    }
}
database.add(sp)
}
}
}
catch (ex:Exception)
{
    ex.printStackTrace()
}
return database
}
}

```

Cuối cùng tới lớp JSON File :



```

package communityuni.com.io

import communityuni.com.model.SanPham
import com.google.gson.Gson
import java.io.FileWriter
import java.io.FileReader
import com.google.gson.reflect.TypeToken

```

```

    /**
     * Created by cafe on 04/06/2017.
     */
    class JSonFileFactory: FileFactory {
        override fun LuuFile(database: MutableList<SanPham>, path: String): Boolean {
            try {
                val gs= Gson()
                val file=FileWriter(path)
                gs.toJson(database,file)
                file.close()
                return true
            }
            catch (ex:Exception)
            {
                ex.printStackTrace()
            }
            return false
        }

        override fun DocFile(path: String): MutableList<SanPham> {
            var database:MutableList<SanPham> = mutableListOf()
            try {
                val gson = Gson()
                var file=FileReader(path)
                database = gson.fromJson<MutableList<SanPham>>(file,
                    object : TypeToken<MutableList<SanPham>>()
                    {
                        type
                    })
                file.close()
            }
            catch (ex:Exception)
            {
                ex.printStackTrace()
            }
            return database
        }
    }

```

Như vậy là bạn đã tạo xong các lớp mô hình, bây giờ ta tiến hành xử lý nghiệp vụ.

vào SanPhamPanel, bổ sung thêm đối tượng để lưu danh sách Sản phẩm, viết lại hàm thêm, hiển thị lên JTable cũng như hiển thị chi tiết:

Trong Lớp SanPhamPanel bổ sung biến:

```
import java.util.ArrayList  
  
var database: List<SanPham> = ArrayList()
```

Sửa lại sự kiện thêm sản phẩm:

```
btnLuu.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        Vector<String> vector=new Vector<>();  
        vector.add(txtMa.getText());  
        vector.add(txtTen.getText());  
        vector.add(txtDonGia.getText());  
        tableModelSanPham.addRow(vector);  
        //bổ sung thêm ở đây:  
        SanPham sp=new SanPham();  
        sp.setMa(txtMa.getText());  
        sp.setTen(txtTen.getText());  
        sp.setGia(Double.parseDouble(txtDonGia.getText()));  
        database.add(sp);  
    }  
});
```

Sửa lại đoạn lệnh cho sự kiện xóa:

```
btnXoa.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        if(tblSanPham.getSelectedRow() >=0)  
        {  
            tableModelSanPham.removeRow(tblSanPham.getSelectedRow());  
            database.remove(tblSanPham.getSelectedRow());  
        }  
    }  
});
```

Bổ sung thêm hàm để hiển thị toàn bộ danh sách Sản phẩm từ biến database lên JTable:

```
public void showDataOnJTable()  
{  
    tableModelSanPham.setRowCount(0);
```

```

    for(int i=0;i<database.size();i++)
    {
        SanPham sp=database.get(i);
        Vector<String>vector=new Vector<>();
        vector.add(sp.getMa());
        vector.add(sp.getTen());
        vector.add(sp.getGia()+"");
        tableModelSanPham.addRow(vector);
    }
}

```

Vậy cuối cùng ta có cấu trúc coding của lớp SanPhamPanel này như sau:

```

package communityuni.com.ui;

import communityuni.com.model.SanPham;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

/**
 * Created by cafe on 04/06/2017.
 */
public class SanPhamPanel {
    private JPanel pnMain;
    private JTable tblSanPham;
    private DefaultTableModel tableModelSanPham;
    private JButton btnLuu;
    private JButton btnTiep;
    private JButton btnXoa;
    private JButton btnThoat;
    private JTextField txtMa;
    private JTextField txtTen;
    private JTextField txtDonGia;

    List<SanPham>database=new ArrayList<>();

    public SanPhamPanel() {
        btnTiep.addActionListener(new ActionListener() {
            @Override

```

```

        public void actionPerformed(ActionEvent e) {
            txtMa.setText("");
            txtTen.setText("");
            txtDonGia.setText("");
            txtMa.requestFocus();
        }
    });
btnThoat.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
btnLuu.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Vector<String>vector=new Vector<>();
        vector.add(txtMa.getText());
        vector.add(txtTen.getText());
        vector.add(txtDonGia.getText());
        tableModelSanPham.addRow(vector);
//bỏ sung thêm ở đây:
        SanPham sp=new SanPham();
        sp.setMa(txtMa.getText());
        sp.setTen(txtTen.getText());

        sp.setGia(Double.parseDouble(txtDonGia.getText()));
        database.add(sp);
    }
});
tblSanPham.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        super.mousePressed(e);
        int row=tblSanPham.getSelectedRow();
        String
ma=tblSanPham.getValueAt(row, 0).toString();
        String
ten=tblSanPham.getValueAt(row, 1).toString();
        double
gia=Double.parseDouble(tblSanPham.getValueAt(row, 2).toString());
        txtMa.setText(ma);
        txtTen.setText(ten);
        txtDonGia.setText(gia+"");
    }
});
tblSanPham.addMouseListener(new MouseAdapter() {

```

```

    });
    btnXoa.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (tblSanPham.getSelectedRow() >= 0)
            {

tableModelSanPham.removeRow(tblSanPham.getSelectedRow());
database.remove(tblSanPham.getSelectedRow());
}
        }
    });
    createICon();
}
public void showDataOnJTable()
{
    tableModelSanPham.setRowCount(0);
    for(int i=0;i<database.size();i++)
    {
        SanPham sp=database.get(i);
        Vector<String>vector=new Vector<>();
        vector.add(sp.getMa());
        vector.add(sp.getTen());
        vector.add(sp.getGia()+"");
        tableModelSanPham.addRow(vector);
    }
}
private void createICon()
{
    btnLuu.setIcon(new ImageIcon("hinh/add.png"));
    btnTiep.setIcon(new ImageIcon("hinh/clear.png"));
    btnXoa.setIcon(new ImageIcon("hinh/delete.png"));
    btnThoat.setIcon(new ImageIcon("hinh/exit.png"));
}
private void createUIComponents()
{
    tableModelSanPham=new DefaultTableModel();
    tableModelSanPham.addColumn("Mã Sản Phẩm");
    tableModelSanPham.addColumn("Tên Sản Phẩm");
    tableModelSanPham.addColumn("Đơn Giá Sản Phẩm");
    tblSanPham=new JTable(tableModelSanPham);
}
public JPanel getPnMain()
{
    return pnMain;
}
}

```

Bây giờ ta vào SanPhamUI để xử lý cho các Menu Lưu và đọc File, bổ sung thêm 2 biến cho FileFactory và JFileChooser, SanPhamPanel:

```
FileFactory fileFactory;
JFileChooser fileChooser=new JFileChooser();
SanPhamPanel sanPhamPanel;
```

Trong Constructor của SanPhamUI sửa lại chỗ setContentPane và gọi hàm createEvent(hàm này do ta tạo thêm để gán sự kiện cho các Menu):

```
public SanPhamUI(String title)
{
    super(title);
    sanPhamPanel=new SanPhamPanel();
    setContentPane(sanPhamPanel.getPnMain());
    createMenu();
    createIcon();
    createEvent();
}

private void createEvent() {
    mnuSystemSaveTextFile.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e) {
            //xử lý lưu TextFile ở đây
        }
    });
}
```

Coding chi tiết cho sự kiện lưu TextFile:

```
private void createEvent() {
    mnuSystemSaveTextFile.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e) {
            //xử lý lưu TextFile ở đây
            fileFactory=new TextFileFactory();

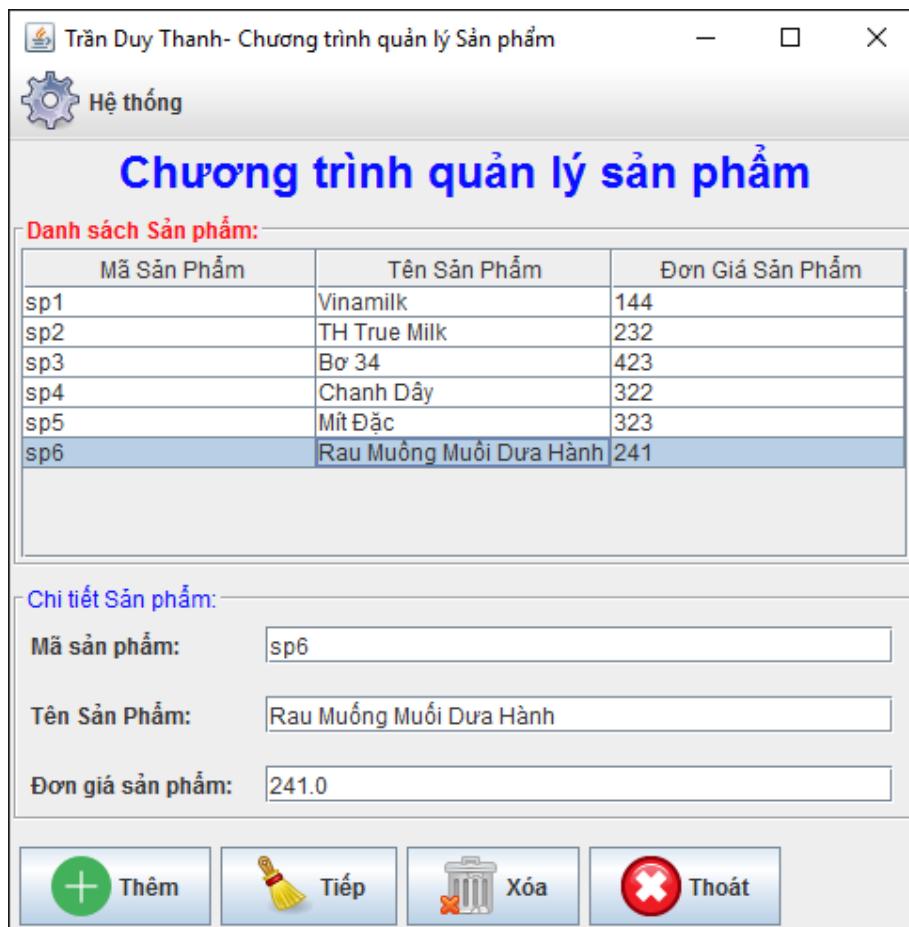
        if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION)
        {
            boolean
```

```

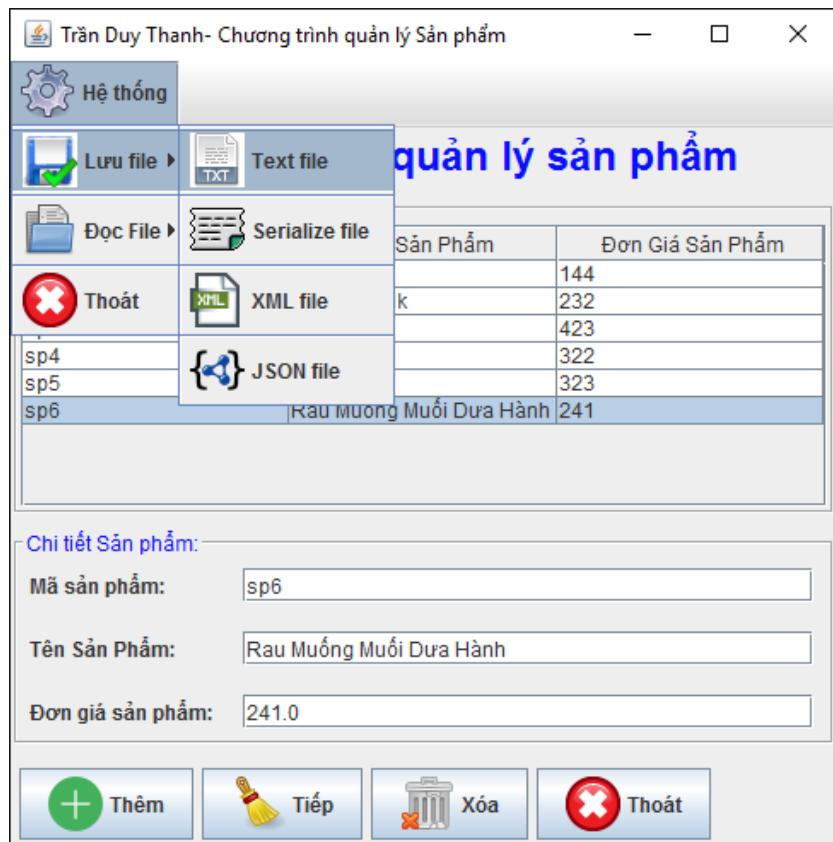
kq=fileFactory.LuuFile(sanPhamPanel.database,fileChooser.getSelectedFile().getAbsolutePath());
        if (kq==true)
        {
            JOptionPane.showMessageDialog(null, "Lưu Text
File thành công");
        }
        else
        {
            JOptionPane.showMessageDialog(null, "Lưu Text
File thất bại");
        }
    }
}
}

```

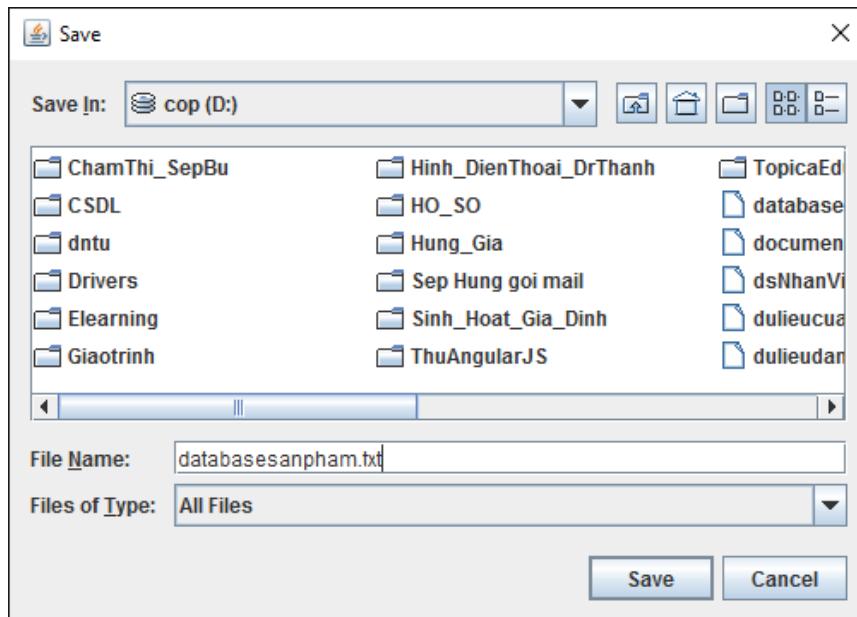
Tới đây ta chạy phần mềm lên để Test chức năng lưu Text File trước xem thành công hay không nhé:



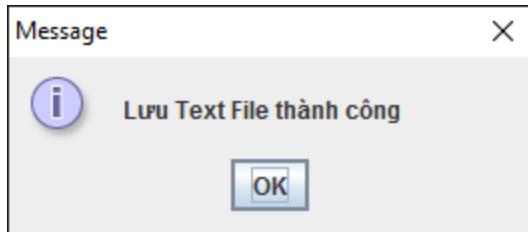
Ta Vào Menu Lưu file/ chọn Save Text File:



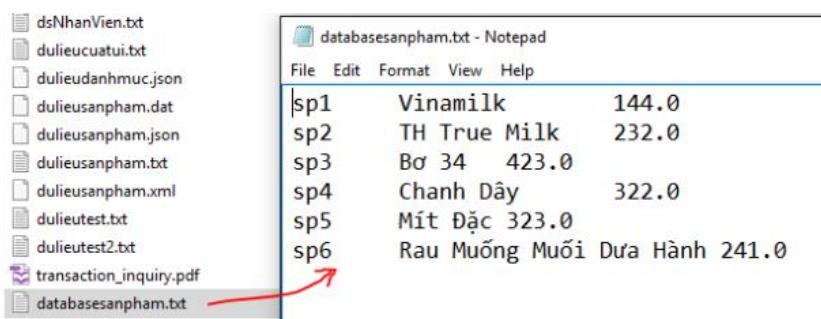
Màn hình JFileChooser sẽ hiển thị ra như dưới đây:



Đặt tên file là databasesanpham.txt (tên gì kệ bạn), chọn nơi lưu trữ bất kỳ(kê bạn). Rồi nhấn nút Save để xác thực lưu Text File, nếu ra cửa sổ thông báo sau là Lưu text file thành công:



Ta kiểm lại xem có đúng không nhé, vào nơi mà ta chọn lưu:

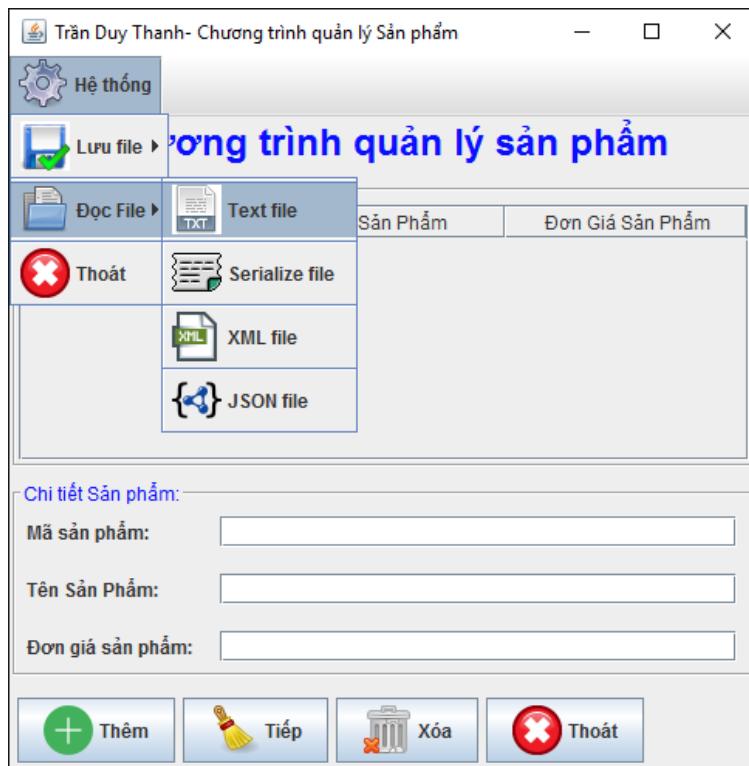


Rõ ràng là lưu Text File thành công, bây giờ ta tắt phần mềm và tiến hành viết lệnh đọc Text File:

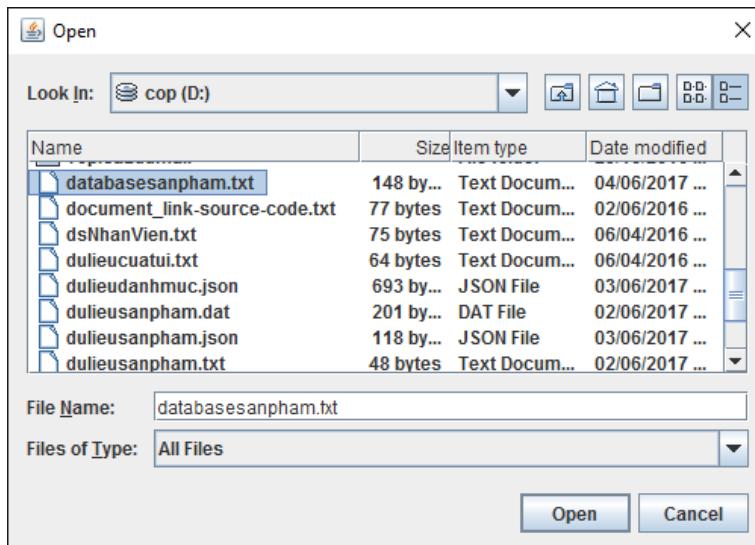
```
private void createEvent() {
    mnuSystemOpenTextFile.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e) {
            fileFactory=new TextFileFactory();

        if(fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION)
        {
            sanPhamPanel.database=fileFactory.DocFile(fileChooser.getSelectedFile().getAbsolutePath());
            sanPhamPanel.showDataOnJTable();
        }
    });
}
```

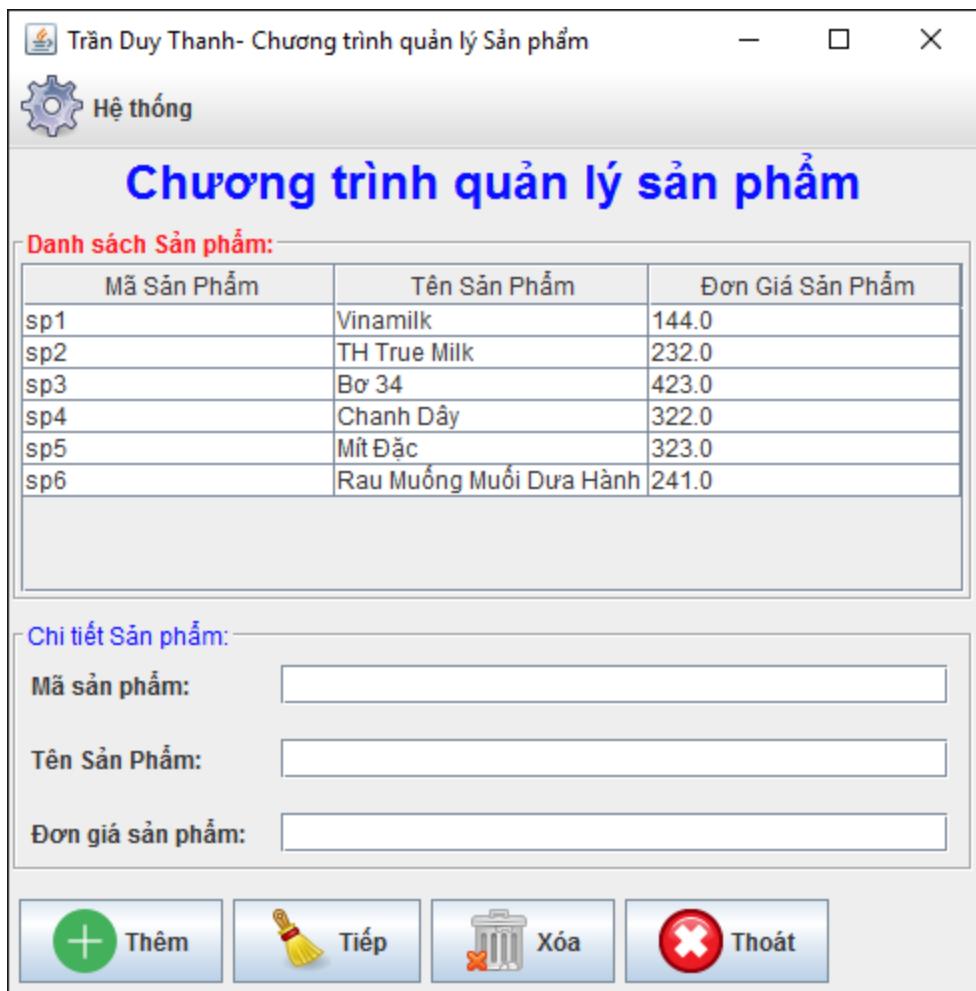
Chạy chương trình lên ta sẽ có giao diện trống rỗng như sau:



Vào menu Hệ thống / chọn Đọc File/ chọn Text File, cửa sổ chọn File xuất hiện:



Chọn đúng tập tin lúc nãy bạn lưu (databasesanpham.txt) rồi bấm Open, kết quả:



Như vậy chúng ta đã Đọc text file thành công, kết thúc công việc lưu file và đọc text file. Còn Serialize, XML, JSON hoàn toàn tương tự, Các bạn tự xử nhé.

và chú ý là không quan tâm lưu dữ liệu với loại định dạng nào, khi đã Nạp lên Bộ nhớ thì ta có thể xuất ra bất kỳ định dạng tập tin nào (4 loại).

Coding đầy đủ của SanPhamUI ở đây:

```

package communityuni.com.ui;

import communityuni.com.io.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/*

```

```

 * Created by cafe on 04/06/2017.
 */
public class SanPhamUI extends JFrame {
    JMenuBar mnuBar;
    JMenu mnuSystem;
    JMenu mnuSystemSave;
    JMenuItem mnuSystemSaveTextFile;
    JMenuItem mnuSystemSaveSerializeFile;
    JMenuItem mnuSystemSaveXMLFile;
    JMenuItem mnuSystemSaveJSonFile;
    JMenu mnuSystemOpen;
    JMenuItem mnuSystemOpenTextFile;
    JMenuItem mnuSystemOpenSerializeFile;
    JMenuItem mnuSystemOpenXMLFile;
    JMenuItem mnuSystemOpenJSonFile;
    JMenuItem mnuSystemExit;

    FileFactory fileFactory;
    JFileChooser fileChooser=new JFileChooser();
    SanPhamPanel sanPhamPanel;
    public SanPhamUI(String title)
    {
        super(title);
        sanPhamPanel=new SanPhamPanel();
        setContentPane(sanPhamPanel.getPnMain());
        createMenu();
        createIcon();
        creatEvent();
    }

    private void creatEvent() {
        mnuSystemOpenTextFile.addActionListener(new
ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                fileFactory=new TextFileFactory();

if(fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION
)
{
    sanPhamPanel.database=fileFactory.DocFile(fileChooser.getSelectedFile().getAbsolutePath());
    sanPhamPanel.showDataOnJTable();
}
}
    }
}

```

```

mnuSystemSaveTextFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //xử lý lưu TextFile ở đây
        fileFactory=new TextFileFactory();

if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION)
{
    boolean
kq=fileFactory.LuuFile(sanPhamPanel.database,fileChooser.getSelectedFile().getAbsolutePath());
    if(kq==true)
    {
        JOptionPane.showMessageDialog(null,"Lưu
Text File thành công");
    }
    else
    {
        JOptionPane.showMessageDialog(null,"Lưu
Text File thất bại");
    }
}
}
);
}

mnuSystemOpenSerializeFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        fileFactory=new SerializeFileFactory();

if(fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION)
{
    sanPhamPanel.database=fileFactory.DocFile(fileChooser.getSelectedFile().getAbsolutePath());
        sanPhamPanel.showDataOnJTable();
}
}
);
}

mnuSystemSaveSerializeFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

```

```

        //xử lý lưu TextFile ở đây
        fileFactory=new SerializeFileFactory();

if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION)
{
    boolean
kq=fileFactory.LuuFile(sanPhamPanel.database,fileChooser.getSelectedFile().getAbsolutePath());
    if(kq==true)
    {
        JOptionPane.showMessageDialog(null,"Lưu
Serialize File thành công");
    }
    else
    {
        JOptionPane.showMessageDialog(null,"Lưu
Serialize File thất bại");
    }
}
}

mnuSystemOpenXMLFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        fileFactory=new XMLFileFactory();

if(fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION)
{
    sanPhamPanel.database=fileFactory.DocFile(fileChooser.getSelectedFile().getAbsolutePath());
        sanPhamPanel.showDataOnJTable();
    }
}
});

mnuSystemSaveXMLFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //xử lý lưu TextFile ở đây
        fileFactory=new XMLFileFactory();

if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION)
}
})

```

```

        {
            boolean
kq=fileFactory.LuuFile(sanPhamPanel.database,fileChooser.getSelectedFile().getAbsolutePath());
            if(kq==true)
            {
                JOptionPane.showMessageDialog(null,"Lưu
XML File thành công");
            }
            else
            {
                JOptionPane.showMessageDialog(null,"Lưu
XML File thất bại");
            }
        }
    });
mnuSystemOpenJsonFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        fileFactory=new JsonFileFactory();

if(fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION)
{
    sanPhamPanel.database=fileFactory.DocFile(fileChooser.getSelectedFile().getAbsolutePath());
    sanPhamPanel.showDataOnJTable();
}
});
mnuSystemSaveJsonFile.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //xử lý lưu TextFile ở đây
        fileFactory=new JsonFileFactory();

if(fileChooser.showSaveDialog(null)==JFileChooser.APPROVE_OPTION)
{
    boolean
kq=fileFactory.LuuFile(sanPhamPanel.database,fileChooser.getSelectedFile().getAbsolutePath());
    if(kq==true)

```

```

        {
            JOptionPane.showMessageDialog(null, "Lưu
Json File thành công");
        }
        else
        {
            JOptionPane.showMessageDialog(null, "Lưu
Json File thất bại");
        }
    }
}

public void createMenu()
{
    mnuBar=new JMenuBar();
    setJMenuBar(mnuBar);
    mnuSystem=new JMenu("Hệ thống");
    mnuBar.add(mnuSystem);

    mnuSystemSave=new JMenu("Lưu file");
    mnuSystem.add(mnuSystemSave);
    mnuSystemSaveTextFile=new JMenuItem("Text file");
    mnuSystemSave.add(mnuSystemSaveTextFile);
    mnuSystemSave.addSeparator();
    mnuSystemSaveSerializeFile=new JMenuItem("Serialize
file");
    mnuSystemSave.add(mnuSystemSaveSerializeFile);
    mnuSystemSave.addSeparator();
    mnuSystemSaveXMLFile=new JMenuItem("XML file");
    mnuSystemSave.add(mnuSystemSaveXMLFile);
    mnuSystemSave.addSeparator();
    mnuSystemSaveJSONFile=new JMenuItem("JSON file");
    mnuSystemSave.add(mnuSystemSaveJSONFile);

    mnuSystem.addSeparator();

    mnuSystemOpen=new JMenu("Đọc File");
    mnuSystem.add(mnuSystemOpen);
    mnuSystemOpenTextFile=new JMenuItem("Text file");
    mnuSystemOpen.add(mnuSystemOpenTextFile);
    mnuSystemOpen.addSeparator();
    mnuSystemOpenSerializeFile=new JMenuItem("Serialize
file");
    mnuSystemOpen.add(mnuSystemOpenSerializeFile);
    mnuSystemOpen.addSeparator();
}

```

```

mnuSystemOpenXMLFile=new JMenuItem("XML file");
mnuSystemOpen.add(mnuSystemOpenXMLFile);
mnuSystemOpen.addSeparator();
mnuSystemOpenJSONFile=new JMenuItem("JSON file");
mnuSystemOpen.add(mnuSystemOpenJSONFile);

mnuSystem.addSeparator();

mnuSystemExit=new JMenuItem("Thoát");
mnuSystem.add(mnuSystemExit);
}

private void createIcon() {
    mnuSystem.setIcon(new ImageIcon("hinh/system.png"));
    mnuSystemSave.setIcon(new ImageIcon("hinh/save.png"));
    mnuSystemSaveTextFile.setIcon(new
ImageIcon("hinh/textfile.png"));
    mnuSystemSaveSerializeFile.setIcon(new
ImageIcon("hinh/serializefile.png"));
    mnuSystemSaveXMLFile.setIcon(new
ImageIcon("hinh/xmlfile.png"));
    mnuSystemSaveJSONFile.setIcon(new
ImageIcon("hinh/jsonfile.png"));

    mnuSystemOpen.setIcon(new ImageIcon("hinh/open.png"));
    mnuSystemOpenTextFile.setIcon(new
ImageIcon("hinh/textfile.png"));
    mnuSystemOpenSerializeFile.setIcon(new
ImageIcon("hinh/serializefile.png"));
    mnuSystemOpenXMLFile.setIcon(new
ImageIcon("hinh/xmlfile.png"));
    mnuSystemOpenJSONFile.setIcon(new
ImageIcon("hinh/jsonfile.png"));

    mnuSystemExit.setIcon(new ImageIcon("hinh/exit.png"));
}
public void showWindow()
{
    setSize(500,500);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setVisible(true);
}
}

```

So sánh định dạng dữ liệu kết quả 4 loại:

databasesanpham.txt - Notepad

```
File Edit Format View Help
sp1    Vinamilk      144.0
sp2    TH True Milk  232.0
sp3    Bơ 34         423.0
sp4    Chanh Dây     322.0
sp5    Mít Đặc       323.0
sp6    Rau Muống Muối Dưa Hành 241.0
```

databasesanpham.dat - Notepad

```
File Edit Format View Help
Í lsr ljava.util.ArrayListxÔ"ÇaÏ Ï
lsizep  lsr
communityuni.com.model.SanPhamP~D<WjÏ
lD lgiaL mat lLjava/lang/String;L ltenq ~
lsp@b t lsp1 Minamilksq ~ @m
t lsp2t þTH True Milksq ~ @zp t
lsp3t lBÆj 34sq ~ @t t lsp4t Chanh
DÃ¢ysq ~ @t0 t lsp5t lMÃ-t Äáº·csq ~
```

databasesanpham.xml - Notepad

```
File Edit Format View Help
standalone="no"?
><SanPhams><SanPham><Ma>sp1</Ma><Ten>Vi
namilk</Ten><Gia>144.0</Gia></SanPham><
SanPham><Ma>sp2</Ma><Ten>TH True
Milk</Ten><Gia>232.0</Gia></SanPham><Sa
nPham><Ma>sp3</Ma><Ten>Bơ
34</Ten><Gia>423.0</Gia></SanPham><SanP
ham><Ma>sp4</Ma><Ten>Chanh
```

databasesanpham.json - Notepad

```
[{"ma": "sp1", "ten": "Vinamilk", "gia": 144.0}, {"ma": "sp2", "ten": "TH True Milk", "gia": 232.0}, {"ma": "sp3", "ten": "Bơ 34", "gia": 423.0}, {"ma": "sp4", "ten": "Chanh Dây", "gia": 322.0}, {"ma": "sp5", "ten": "Mít Đặc", "gia": 323.0}]
```

Như vậy Tui đã hướng dẫn xong toàn bộ phần mềm quản lý sản phẩm theo HUỐNG ĐỐI TUỢNG, thao tác với các control cơ bản và nâng của trong Kotlin. Kết xuất dữ liệu ra 4 loại tập tin, dễ dàng triệu gọi coding giữa Kotlin vs Java rất hay. Bài Sau Tui sẽ làm minh họa về JTree trong Kotlin, các bạn chú ý theo dõi nhé.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/1wbbj8tb05vtvqe/HocGUIPhan4.rar>

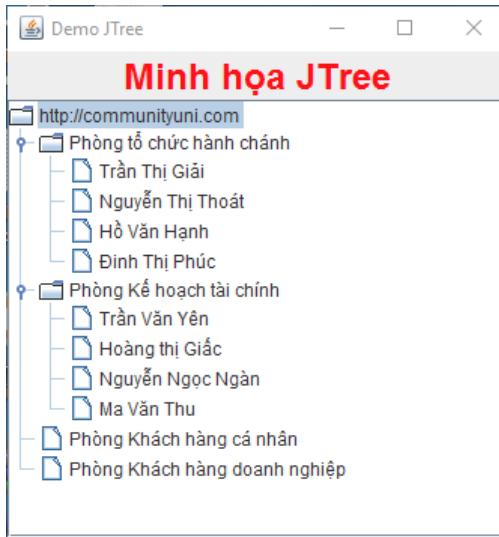
Hẹn gặp các bạn ở những bài tiếp theo

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

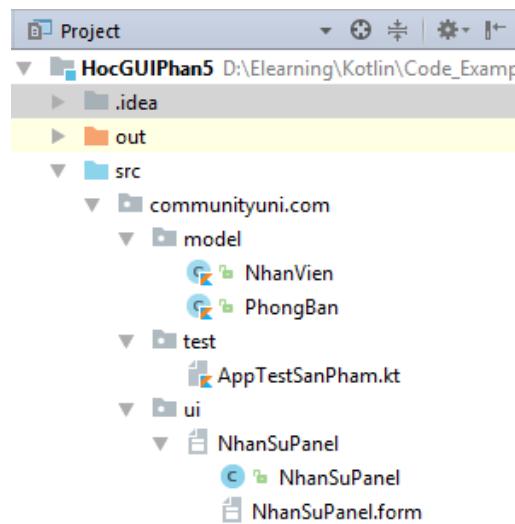
Bài 39-Thiết kế giao diện trong Kotlin – phần 5

Chúc mừng các bạn đã vượt qua 4 cửa ải GUI trong Kotlin, đây là cửa ải cuối cùng Tui muốn các bạn vượt qua. Phần này Tui sẽ nói về JTree, một trong những Control phổ biến được dùng rất nhiều trong các dự án để hiển thị dữ liệu dạng phân cấp cây thư mục:



Bài học này các bạn sẽ biết cách tạo 1 JTree, đưa được DefaultMutableTreeNode vào JTree. Đặc biệt sử dụng lập trình Hướng Đối Tượng để đưa dữ liệu lên Cây cối này, cũng như biết cách xử lý sự kiện khi người dùng nhấp vào từng Node trên Cây.

Bạn tạo một Project có cấu trúc như sau:



Lớp Nhân Viên (NhanVien) có cấu trúc (Mã, tên):

```
package communityuni.com.model

/**
 * Created by cafe on 04/06/2017.
 */
class NhanVien {
    var Ma:Int=0
    var Ten:String=""
    constructor()
    constructor(Ma: Int, Ten: String) {
        this.Ma = Ma
        this.Ten = Ten
    }

    override fun toString(): String {
        return Ten
    }
}
```

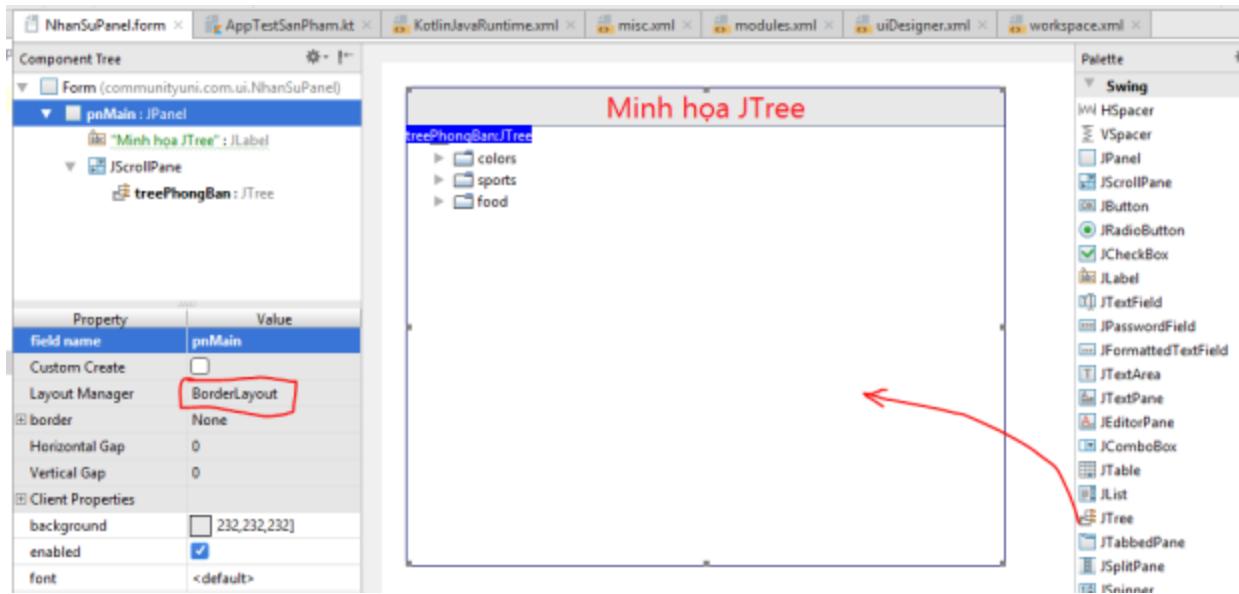
Lớp Phòng Ban (PhongBan) có cấu trúc (mã, tên, danh sách nhân viên):

```
package communityuni.com.model

/**
 * Created by cafe on 04/06/2017.
 */
class PhongBan {
    var Ma:Int=0
    var Ten:String=""
    var NhanViens:MutableList<NhanVien> = mutableListOf()
    constructor()
    constructor(Ma: Int, Ten: String) {
        this.Ma = Ma
        this.Ten = Ten
    }

    override fun toString(): String {
        return Ten
    }
}
```

Tiếp theo thiết kế màn hình **NhanSuPanel.form** như hình dưới:



Chỉnh pnMain có layout manager là BorderLayout, kéo JScrollPane vào phần Center
=> sau đó kéo JTree vào bên trong JScrollPane.

Nhớ chọn JTree rồi checked Custom Create để nó ra hàm createUIComponents()

Sau đó chỉnh sửa source code của NhanSuPanel, giả lập một số dữ liệu:

```

package communityuni.com.ui;

import communityuni.com.model.NhanVien;
import communityuni.com.model.PhongBan;

import javax.swing.*;

import javax.swing.tree.DefaultMutableTreeNode;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by cafe on 04/06/2017.
 */
public class NhanSuPanel {
    private JPanel pnMain;
    private JTree treePhongBan;
    DefaultMutableTreeNode root;
    List<PhongBan> database=null;

    private void loadSampleDatabaseToGUI()
    {

```

```

        root=new
DefaultMutableTreeNode ("http://communityuni.com");
        treePhongBan=new JTree (root);
        for (int i=0;i<database.size();i++)
{
    PhongBan pb=database.get(i);
    DefaultMutableTreeNode pbNode=new
DefaultMutableTreeNode (pb);
    root.add(pbNode);
    for (int j=0;j<pb.getNhanViens ().size();j++)
    {
        NhanVien nv=pb.getNhanViens () .get(j);
        DefaultMutableTreeNode nvNode=new
DefaultMutableTreeNode (nv);

        pbNode.add(nvNode);
    }
}
private void createSampleDatabase ()
{
    database=new ArrayList<> ();
    PhongBan pns=new PhongBan(1,"Phòng tổ chức hành chánh");
    PhongBan phc=new PhongBan(2,"Phòng Kế hoạch tài chính");
    PhongBan pkhcn=new PhongBan(3,"Phòng Khách hàng cá
nhân");
    PhongBan pkhdn=new PhongBan(4,"Phòng Khách hàng doanh
nghiệp");

    database.add(pns);database.add(phc);database.add(pkhcn);database
.add(pkhdn);
    pns.getNhanViens ().add(new NhanVien(1,"Trần Thị Giải"));
    pns.getNhanViens ().add(new NhanVien(2,"Nguyễn Thị
Thoát"));
    pns.getNhanViens ().add(new NhanVien(3,"Hồ Văn Hạnh"));
    pns.getNhanViens ().add(new NhanVien(4,"Đinh Thị Phúc"));
    phc.getNhanViens ().add(new NhanVien(5,"Trần Văn Yên"));
    phc.getNhanViens ().add(new NhanVien(6,"Hoàng thị
Giáć"));
    phc.getNhanViens ().add(new NhanVien(7,"Nguyễn Ngọc
Ngàn"));
    phc.getNhanViens ().add(new NhanVien(8,"Ma Văn Thu"));
}

public JPanel getPnMain ()
{
    return pnMain;
}

```

```

    }

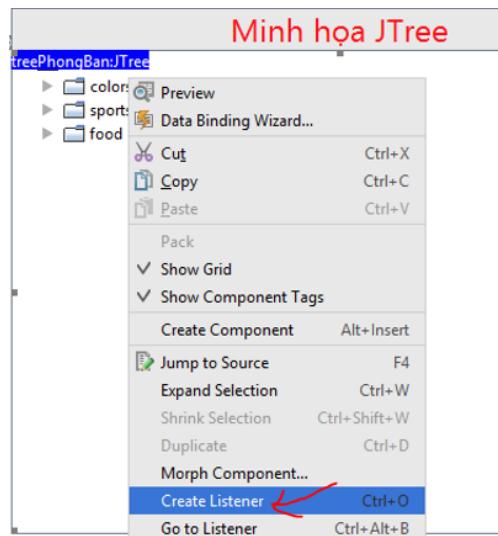
private void createUIComponents() {
    createSampleDatabase();
    loadSampleDatabaseToGUI();
}

}

```

Chạy chương trình lên ta sẽ có giao diện như Tui cung cấp ở trên.

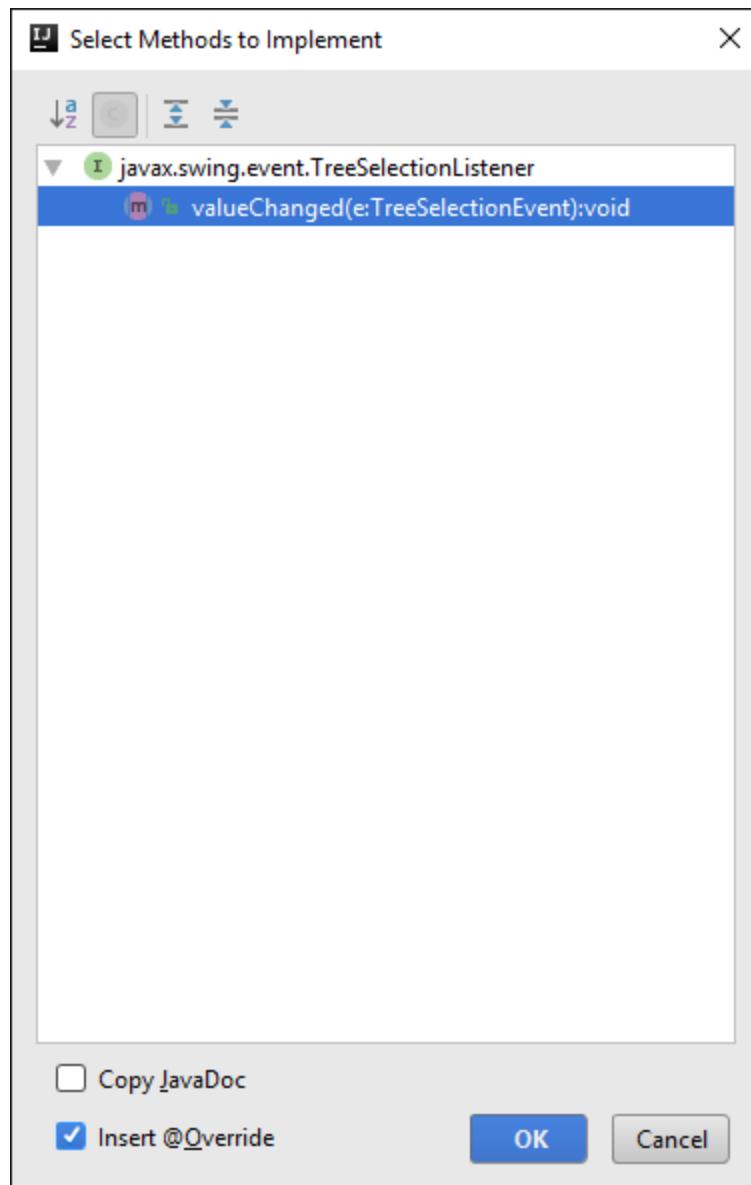
Bây giờ để gán sự kiện cho JTree ta bấm chuột phải vào JTree rồi chọn Create Listener:



Sau đó chọn TreeSelectionListener:



Chọn valueChanged trong màn hình Select Methods to Implement:



Nhấn OK ta thấy sự kiện xuất hiện:

```
public NhanSuPanel() {
    treePhongBan.addTreeSelectionListener(new
TreeSelectionListener() {
    @Override
    public void valueChanged(TreeSelectionEvent e) {
        }
    });
}
```

Bổ sung coding để kiểm tra Node nào được nhấn:

```
public NhanSuPanel() {
    treePhongBan.addTreeSelectionListener(new
TreeSelectionListener() {
    @Override
    public void valueChanged(TreeSelectionEvent e) {
        DefaultMutableTreeNode node=
(DefaultMutableTreeNode)
treePhongBan.getLastSelectedPathComponent();
        switch (node.getLevel())
        {
            case 0:
                JOptionPane.showMessageDialog(null, "Bạn nhấn
Root=" + node.getUserObject());
                break;
            case 1:
                PhongBan pb= (PhongBan) node.getUserObject();
                JOptionPane.showMessageDialog(null, pb.getTen());
                break;
            case 2:
                NhanVien nv= (NhanVien) node.getUserObject();
                JOptionPane.showMessageDialog(null, nv.getTen());
                break;
        }
    }
});
```

Như vậy ta dùng treePhongBan.getLastSelectedPathComponent(); để biết được Node nào được chọn, dùng node.getLevel() để lấy chính xác cấp (thực ra là lấy đối tượng), ứng với mỗi level ta sẽ kiểm tra để lấy Đối tượng.



Như vậy Tui đã hướng dẫn xong cách dùng JTree trong Kotlin. các bạn nhớ kế hợp nó với JTable để hiển thị chi tiết dữ liệu.

Các bạn có thể tải source code bài này ở đây:

<http://www.mediafire.com/file/goj6bdoxe2h3bq/HocGUIPhan5.rar>

Hẹn gặp các bạn ở cuối cùng, bài 40 của khóa học Kotlin này.

Chúc các bạn thành công!

Trần Duy Thành (<http://ssoftinc.com/>)

Bài 40-Kết xuất Executable cho Kotlin [Kết thúc khóa học Kotlin]

Chào các bạn!

Chúng ta Say Goodbye ngôn ngữ lập trình Kotlin ở đây nhé, Tui phải Busy cho nhiều tasks khác. Toàn bộ các bài giảng về Kotlin Tui đã tổng hợp trong link <https://duythanhcse.wordpress.com/kotlin/kotlin-co-ban-den-nang-cao/> các bạn vào đây học nhé. Ráng học cho tốt vì sắp tới Android Studio 3.0 ra đời nó sẽ đánh kém lập trình Kotlin For Android, Tui gọi tắt là KfA. Nếu bạn học tới bài 40 này tức là đã đi trước nhiều người một số bước rồi.

Theo Tui tham khảo từ nhiều nguồn, thì trong tương lai sẽ bùng nổ các dự án liên quan tới Machine Learning (Máy học). Có nhiều ngôn ngữ để lập trình cho Machine Learning chẳng hạn như Python, R, Matlab... Nhiều Đại Học lớn trên thế giới hiện nay đã dùng ngôn ngữ lập trình Python vào giảng dạy kỹ thuật lập trình thay thế cho C++ và Java... Trước đó Tui có viết được một số bài về Python <https://duythanhcse.wordpress.com/python/> nhưng do Busy quá phải tạm Pause, Nếu các bạn còn trẻ, khỏe, nhanh nhẹn(xấu trai cũng được) thì hãy tiếp tục nghiên cứu Python, R để về sau có nhiều cơ hội tham gia các dự án Machine Learning.

OK OK OK

Giờ Tui kết thúc khóa học Kotlin với bài hướng dẫn cách Kết xuất Kotlin ra Jar file để người sử dụng có thể bấm 1 phát là chạy luôn mà không cần mở công cụ lập trình.

IntelliZ IDEA cung cấp sẵn cho chúng ta Tool để làm điều này, vô cùng đơn giản đến mức chúng ta không thể nghĩ ra được. Cách làm chi tiết như sau:

1)Bước 1: Chọn 1 Project nào đó, trong hướng dẫn này Tui chọn bài Quản Lý Sản Phẩm tại link <https://duythanhcse.wordpress.com/2017/06/04/bai-38-thiet-ke-giao-dien-trong-kotlin-phan-4/>

Tui hướng dẫn thêm 1 cách tạo class có chứa hàm main để chạy:

-Cách Cũ chúng ta đang làm (tập tin AppTestSanPhamUI.kt):

```

package communityuni.com.test

import communityuni.com.ui.SanPhamUI

/*
 * Created by cafe on 04/06/2017.
 */

fun main(args: Array<String>) {
    var gui:SanPhamUI= SanPhamUI("Trần Duy Thanh- Chương trình
quản lý Sản phẩm")
    gui.showWindow()
}

```

-Cách mới Ta có thể tạo thành một lớp theo cấu trúc dùng **companion object** và notation **@JvmStatic**:

```

package communityuni.com.test

import communityuni.com.ui.SanPhamUI

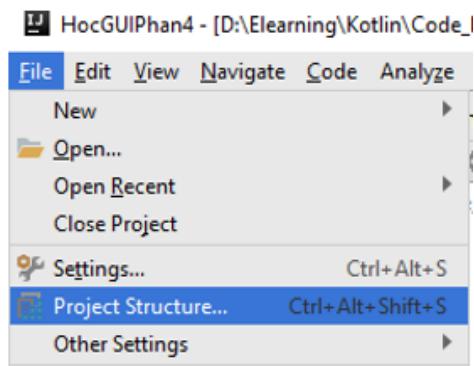
/*
 * Created by cafe on 06/06/2017.
 */

class SanPhamApp {
    companion object
    {
        @JvmStatic
        fun main(args: Array<String>) {
            var gui: SanPhamUI = SanPhamUI("Trần Duy Thanh-
Chương trình quản lý Sản phẩm")
            gui.showWindow()
        }
    }
}

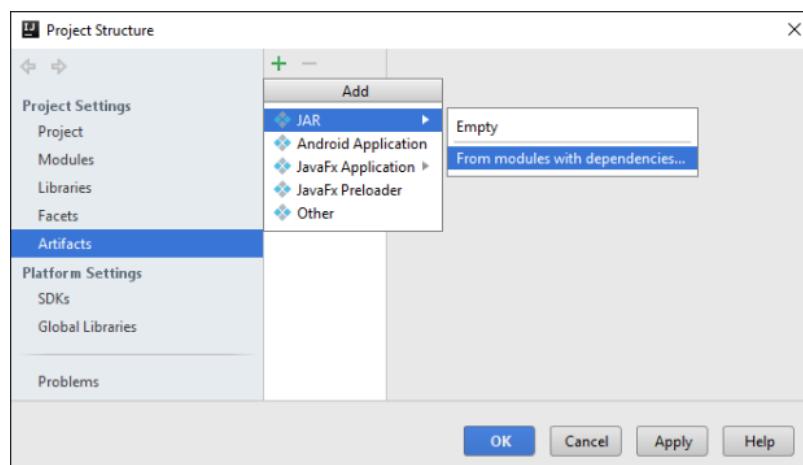
```

Vì sao lại phải biết thêm cách mới này để chạy hàm main? Bạn cứ làm nhiều đi rồi sẽ
biết (**thiên cơ bất khả lộ**)

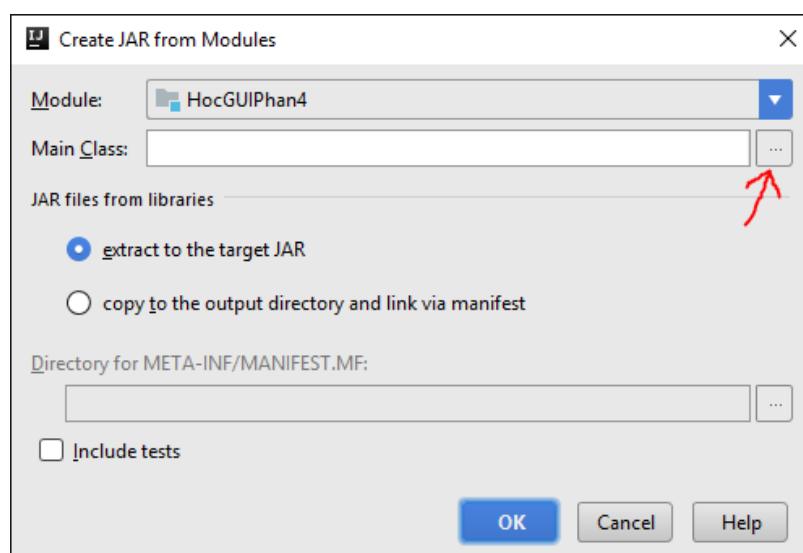
2)Bước 2: Vào File/ chọn Project Structure:



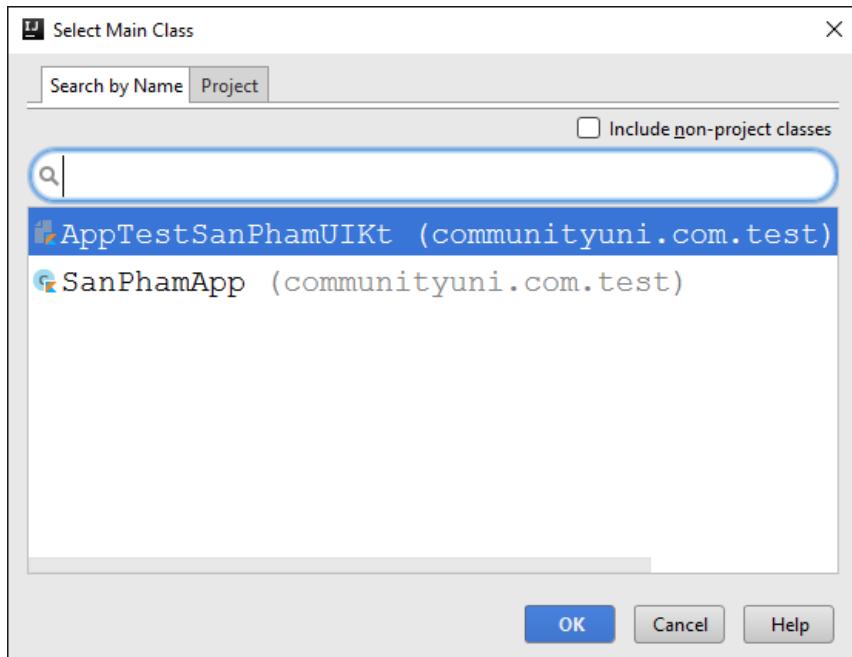
3)Bước 3: Chọn artifacts/ bấm vào dấu + màu xanh / chọn JAR/ chọn From Modules with dependencies...



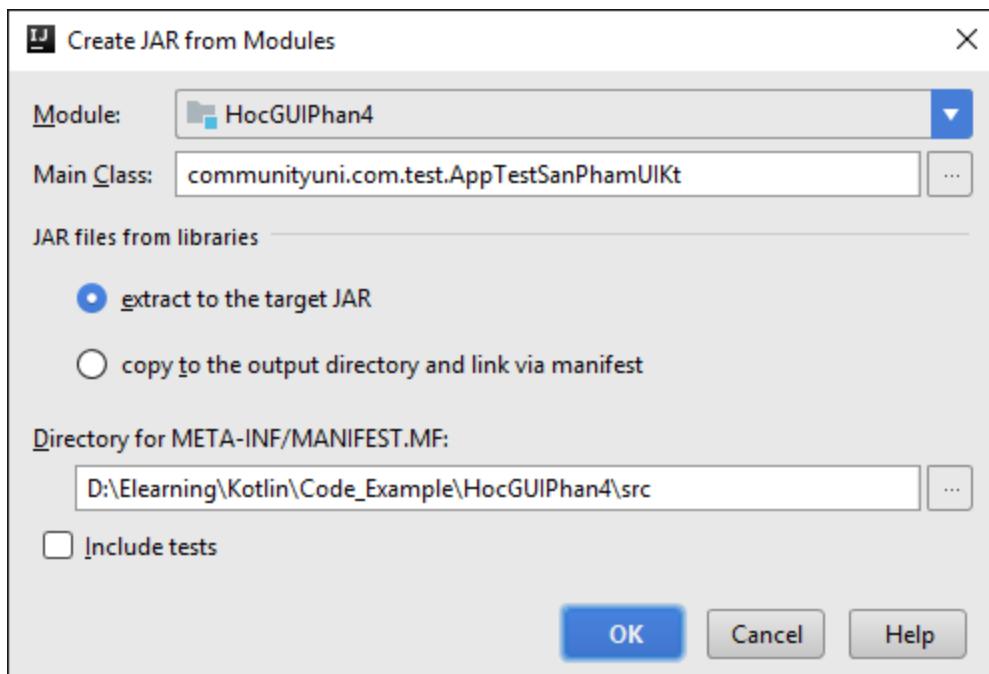
Lúc này màn hình yêu cầu chọn Main Class xuất hiện:



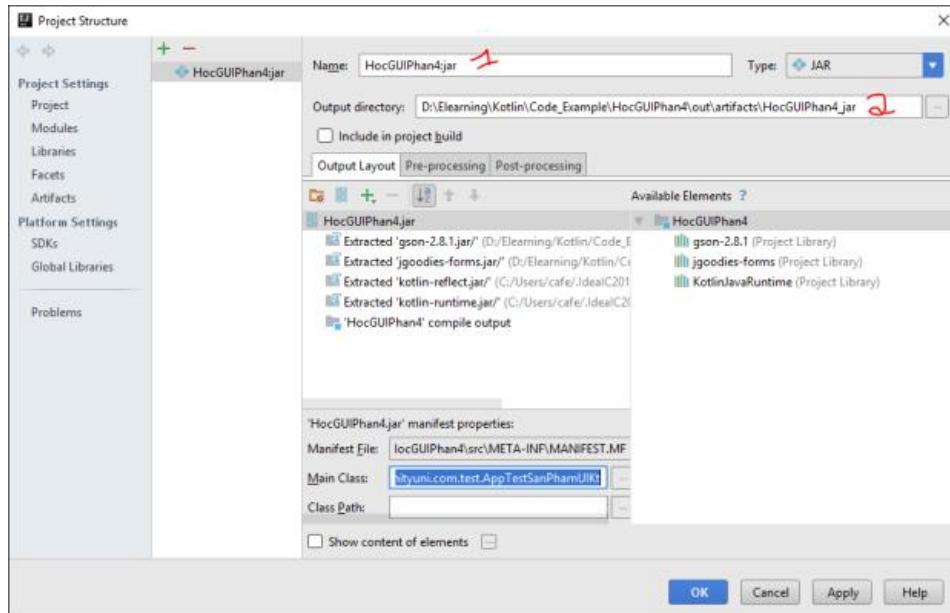
Ta bấm vào nút chọn Main Class:



ở trên bạn chọn cái nào cũng được (2 cách tạo hàm main mà Tui trình bày ở trên) rồi nhấn nút OK

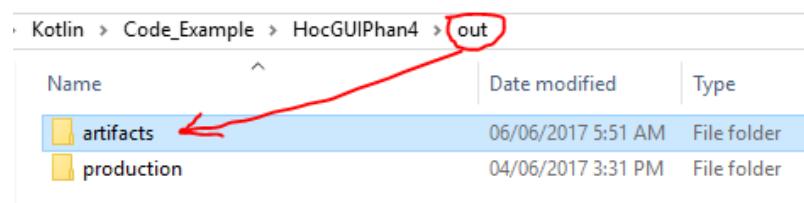


Bấm OK tiếp:

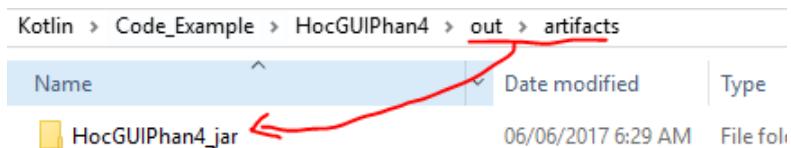


Mục 1 là tên jar được tạo ra, mục 2 là nơi lưu trữ Jar này. Ta chọn OK để quay lại màn hình chính

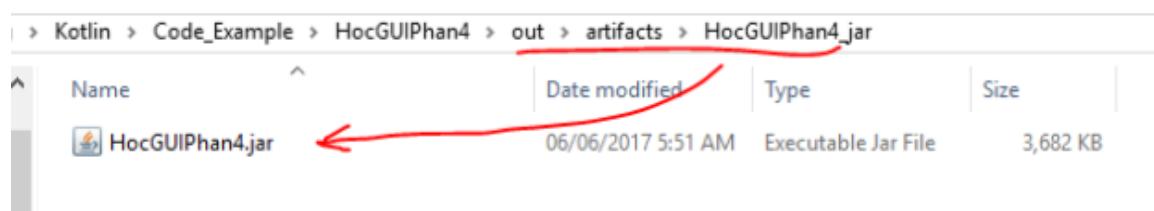
Tiến hành chạy lại phần mềm ta thấy nó tạo ra thư mục artifacts trong out folder:



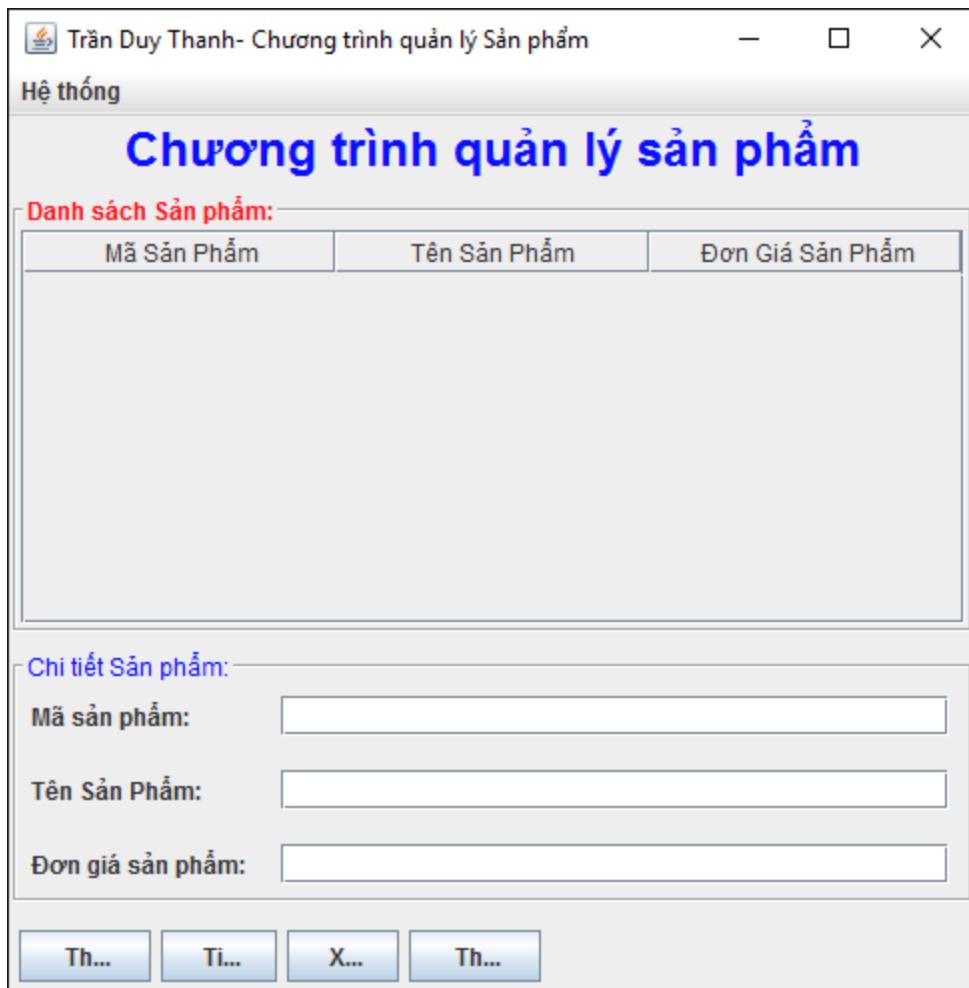
Vào bên trong artifacts có thư mục lưu file jar đó là HocGUIPhan4.jar:

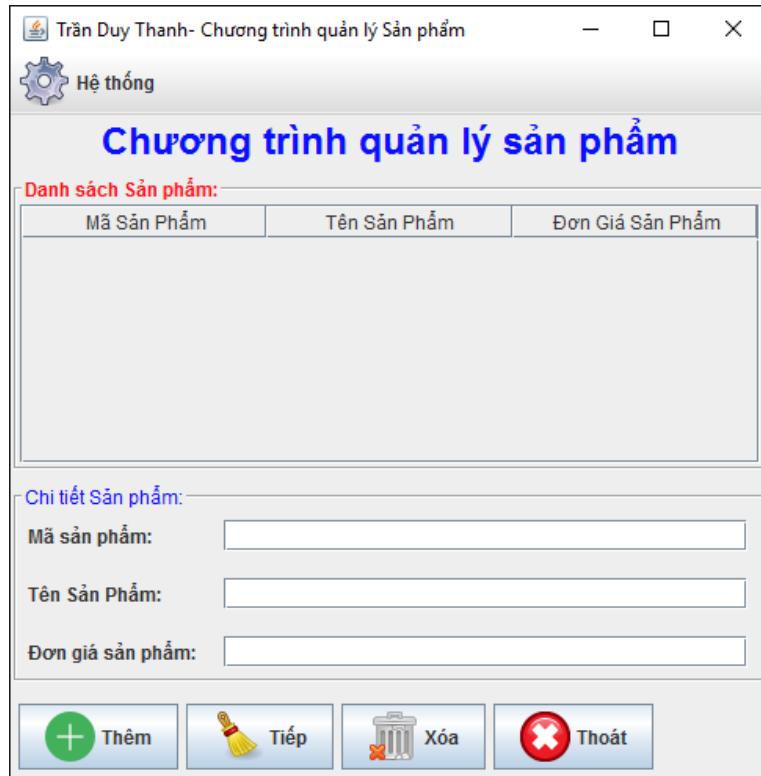


Vào bên trong thư mục này ta thấy file jar:

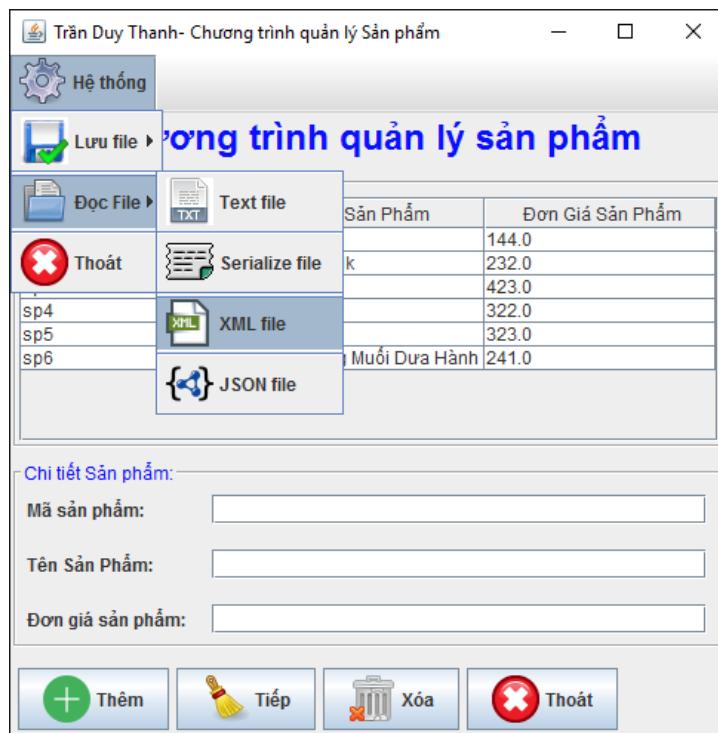


Bạn double click để chạy nó lên:





Hình ảnh đã được hiển thị lên==> Ngon như cơm mẹ nấu đúng không các bạn
Giờ thử vào Menu Hệ thống/ chọn mở File bất kỳ xem nó hiển thị được không nhé:



Như vậy là đã tải file thành công.

Bạn đã hoàn thành các bước để tạo file Jar trong IntelliJ IDEA.

Các bạn cố gắng học tốt các ngôn ngữ lập trình nhé, hãy tự đào tạo mình để có nhiều kiến thức về công nghệ. Tạo ra nhiều cơ hội trong tương lai để tìm tới những Công ty có cơ hội làm việc tốt hơn, lương bổng ổn định hơn.

Để học lập trình tốt các bạn phải chịu khó cày, học ngày học đêm và bắt buộc phải Practice thật nhiều.

Chúc các bạn thành công

Trần Duy Thành (<http://ssoftinc.com/>)

Tài liệu tham khảo

1. <http://kotlinlang.org/>
2. Kotlin in Action
3. Fundamental Kotlin
4. Programming Kotlin
5. Kotlin for Android Developers
6. Modern Web Development with Kotlin

---HẾT---