**Câu 1: Java là ngôn ngữ thông dịch hay biên dịch? Vì sao ?**

* Trình biên dịch (Compiler):

+ Là phương pháp chuyển đổi toàn bộ ngôn ngữ sang mã máy rồi chứa vào ổ cứng để có thể thực hiện chạy lần sau. Ngôn ngữ cấp cao được chuyển đổi gọi là ngôn ngữ mã nguồn (course program) , chương trình ngôn ngữ máy tạo ra là chương trình đối tượng(Object program) hoặc mã đối tương( Object code)

* Trình thông dịch (interpreter):

+ Khi nào chạy chương trình thì thông dịch mới hoạt động để đưa ra mã máy và hiện kết quả

+ **Ưu điểm**: Lập trình viên vẫn chạy được chương trình đến khi có lỗi sảy ra, lỗi tới đâu thì sẽ có thông báo và dừng lại

+ **Nhược điểm:** interpreter chậm hơn compiler vì mỗi lần chạy chương trình đều phải dịch sang mã máy rồi mới chạy.

* Ngôn ngữ java là ngôn ngữ sử dụng trình biên dịch (compiler) , và có máy ảo JVM( java virtual machine) để thông dịch đồng thời khi chạy chương trình. Vì :
* Khi nhận mã nguồn => Java compiler biên dịch từ file.**java** sang file.**class** có chứa bytecode mà máy ảo JVM hiểu.
* Khi nhận được bytecode JVM sẽ thông dịch từng dòng và chạy trên trên máy. Nếu có lỗi chương trình sẽ dừng lại và thông báo.

Note: Write once run anywhere ( viết 1 lần chạy mọi nơi) .

**Câu 2 : Sự khác nhau giữa JDK ,JRE,JVM ?**

* **JVM (Java virtual machine): Máy ảo**

+ Là máy ảo cung cấp môi trường runtime mà bytecode có thể thực thi (chương trình nào, ngôn ngữ nào cũng được biên dịch là java bytecode).

+ Có sẵn cho nhiều nền tảng phần cứng và phần mềm .

+ Phụ thuộc vào nền tảng vì cấu hình mỗi HĐH khác nhau. Tuy nhiên Java là nền tảng động lập. Có 3 khái niệm về JVM : Specification (Chỉ rõ) , implementation (Thi hành), instance ( Ví dụ chứng minh).

+ JVM có 4 nhiệm vụ chính:

1:Nhận mã (Loads code)

2:Kiểm tra (Verifies code)

3:Thực hiện (Executes code)

4:Cung cấp môi trường chạy chương trình (Provides runtime eniroment)

* **JRE( Java Runtime Environment) hay Java RTE:Môi trường chạy ctrinh**

+ Môi trường thực thi Java(JRE) là tập hợp các công cụ phần mềm được sử dụng để phát triển các ứng dụng java.Có tồn tại về mặt vật lí

+ Được sử dụng để cung cấp môi trường thực thi (là việc thực hiện JVM ). Nó tồn tại về mặt vật lý. Chứa tập hợp các thư viện và các tệp khác mà JVM sử dụng trong thời gian thực thi

* **JDK(Java development kit): Phát triển phần mềm**

+Là môi trường phát triển phần mềm được sử dụng để phát triển ứng dụng Java. Có tồn tại về mặt vật lí.

+Có chứa các công cụ phát triểm JRE+

+Gần như là bắt buộc nếu muốn chạy trên nên tảng do Tập đoàn Oracle ( ứng dụng phần mềm Java) phát hành.

+ JDK chứa các máy apr java riêng(JVM) và 1 số tài nguyên khác như trình thông dịch(java), trình biên dịch (javac), trình lưu giữ(jar), trình tài liệu (Javadoc),… để phát triển ứng dụng

* JDK bao gồm (JRE bao gồm JVM).

**Câu 3: Phân biệt abstract và interface**

|  |  |
| --- | --- |
| Abstract class | Abstract Interface |
| Abstract class có thể chứa phương thức absctract và non-abstract | Interface chỉ chứa abstract |
| K hỗ trợ đa kế thừa | Hỗ trợ đa kế thừa |
| Có biến final , static, non-static | Chỉ có static và final |
| Có **phương thức** Static,main,constructor | Không có **phương thức** |
| Từ khóa abstract để khai báo lớp trừ tương | Từ khóa intereface để khai báo interface |
| Tạo được phương thức để triển khai tiến trình | Không tạo được phương thức, chỉ tạo được thuộc tính cho đối tượng |

**Câu 4: Contructor là gì? Hiểu biết của e về contructor**

* KN: Contructor trong java là một dạng đặc biệt của phương thức được sử dụng để khởi tạo đối tượng. Nó được gọi tại thời điểm tạo đối tượng, nó khởi tạo các dữ liệu cho các đối tượng.
* Quy tắc cơ bản tạo contrucctor

+ Tên contructor phải giống tên lớp của nó

+Contrucor không có kiểu trả về tường minh

* Các kiểu của java contructor

+ Mặc định : Contructor mặc định cung cấp các giá trị mặc định như 0, null ,… tùy thuộc vào kiểu dữ liệu với đối tượng được khởi tạo.

+Tham số : Là contructor có tham số truyền vào.Tham số được sử dụng để cung cấp các giá trị khác nhau cho các đối tượng khác nhau

* Contructor overloading(nạp chồng) : Có thể tạo nhiều contructor khác nhau thông qua số lượng và kiểu của tham số truyền vào
* Sự khác nhau giữa contructor và method

|  |  |
| --- | --- |
| **Contructor** | **Method** |
| Được sử dụng để khởi tạo trạng thái của 1 đối tượng | Được sử dụng để thể hiện hành động của 1 đối tượng |
| Không có kiểu trả về | Có kiểu trả về |
| Được gọi là ngầm | Được gọi là tường minh |
| Nếu k tự tạo sẽ đc trình biên dịch tạo mặc định | Coder phải tự tạo.Trình biên dịch k tự tạo |
| Tên phải giống tên class | Tên có thể giống hoặc khác của class |

**Câu 5: Các từ khóa new,static,this,super,final**

* **New :**

+ Là 1 từ khóa (keyword)

+ Dùng cấp phát bộ nhớ cho đối tượng

* **Static :**

+ Để quản lí bộ nhớ

+ Áp dụng với các biến , phương thức, khối, các lớp lồng nhau .Từ khóa static thuộc về lớp chứ k thuộc về instance(thực thể) của lớp.

+Trong java , static có thể là:

1:Biến static: Khi bạn khai báo là static thì đó là biến tĩnh

+ Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng).VD: tên công ti của nhân viên, tên trường của sinh viên,…

+BIến static lấy bộ nhớ chỉ 1 lần duy nhất trong classArea tại thời gian tải lớp đó.

+ **Lợi thế** : Tiết kiệm bộ nhớ

2:Phương thức static :Khi bạn khi báo 1 phương thức là static , thì phương thức đó gọi là phương thức static:

+ Một phương thức static thuộc lớp chứ không phải đối tượng của lớp

+ Một phương thức static gọi mà không cần tạo một instance của một lớp. + Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó.

+**Nhược điểm:**

* Không thể sử dụng biến non-static hay gọi trực tiếp phương thức non-static
* Từ khóa this và supper không sử dụng trong ngữ cảnh static

3:Khối static: Được sử dụng để khởi tạo thành viên dữ liệu static .

+ Được sử dụng dụng để khởi tạo thành viên dữ liệu static

+Được thực thi trước hàm main

* **Từ khóa this:** Có 6 cách sử dụng

1. Từ khóa this tham chiếu tới biến instance của lớp hiện tại :Nếu tên biến toàn cục và tham số trùng nhau
2. Phương thức this() dùng để gọi contructor của lớp hiện tại: this() phải được khai báo dòng lệnh đầu tiên trong contructor.
3. Gọi phưng thức bằng từ khóa this
4. Sử dụng từ khóa this như 1 tham số của phương thức (CÁI NÀY KHÔNG HIỂU)
5. Sử dụng từ khóa this như 1 tham số của contructor (CÁI NÀY KHÔNG HIỂU)
6. Sử dụng this để trả về instance của lớp hiện tại

* **Từ khóa Super** :

+ Sử dụng tham chiếu trực tiếp tới biến instance của lớp cha

+ Sử dụng super() để gọi trực tiếp contructor của lớp cha

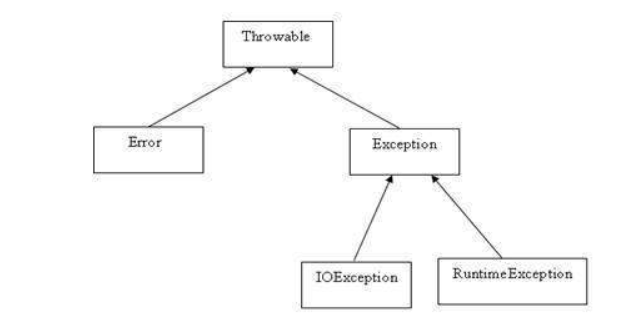
+ Sử dụng để gọi trực tiếp phương thức của lớp cha

* **Từ khóa final** :

Được sử dụng để hạn chế người dùng:

1. Biến final là hằng số
2. Phương thức final k thể ghi đè
3. Lớp final k thể được kế thừa
4. Biến final mà không được khởi tjao tại thời điểm khai báo gọi là biến final trống.

**Câu 6:Hiểu biết về Exception trong java**

****

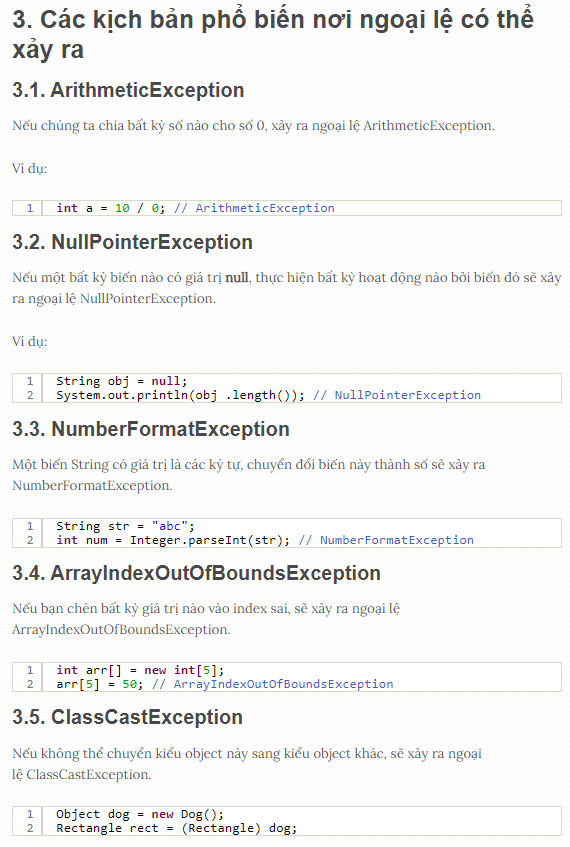
Exception là một tình trạng bất thường , là sự kiện phá vỡ luồng chuẩn của chương trình. Nó là đối tượng ném tại run -time .

Exception là vấn đề sảy ra trong quá trình thực hiện chương trình. có 2 loại

1: Checked E :Là ngoại lệ k thể lường trước được của lập trình viên

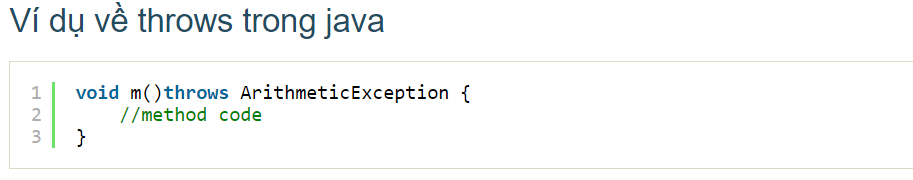
2: Unchecked E : Là loại sảy ra khi chạy chương trình (run -time ) , có thể tránh được bởi lập trình viên

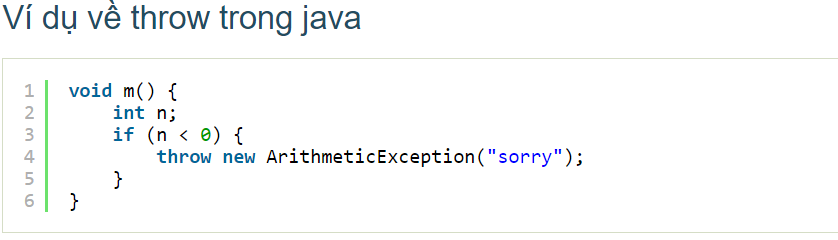
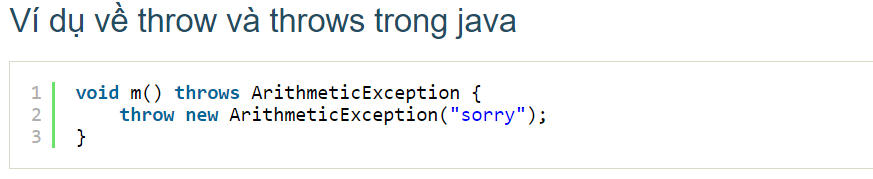
3:Error : k giống E , sảy ra khi lỗi say k thể kiểm soát bởi lập trình viên hay người dùng => Bỏ qua



**Câu 7:Phân biệt throw và throw**

|  |  |
| --- | --- |
| Throws | Throw |
| Để khai báo ngoại lệ | Được sử dụng để ném ra ngoại lệ rõ ràng |
| Ngoại lệ checked được truyền ra ngay cả khi dùng Throws | Ngoại lệ checked k được truyền ra nếu chỉ sử dụng throw |
| Sau Throws là 1 hoặc nhiều class | Sau throw là 1 instance |
| Được khai báo ngay khi đóng dấy ngược đơn của phương thức | Throw được sử dụng trong phương thức |
| Có thể khai báo nhiều exception | Không thể throw nhiều exception |



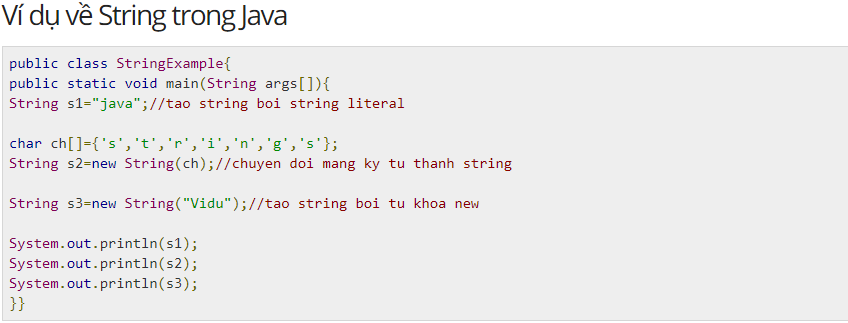


**Câu 8: String , Stringbuider , StringBuffer** ( Nói thật là phần này cũng k hiểu lắm nhất là builder)

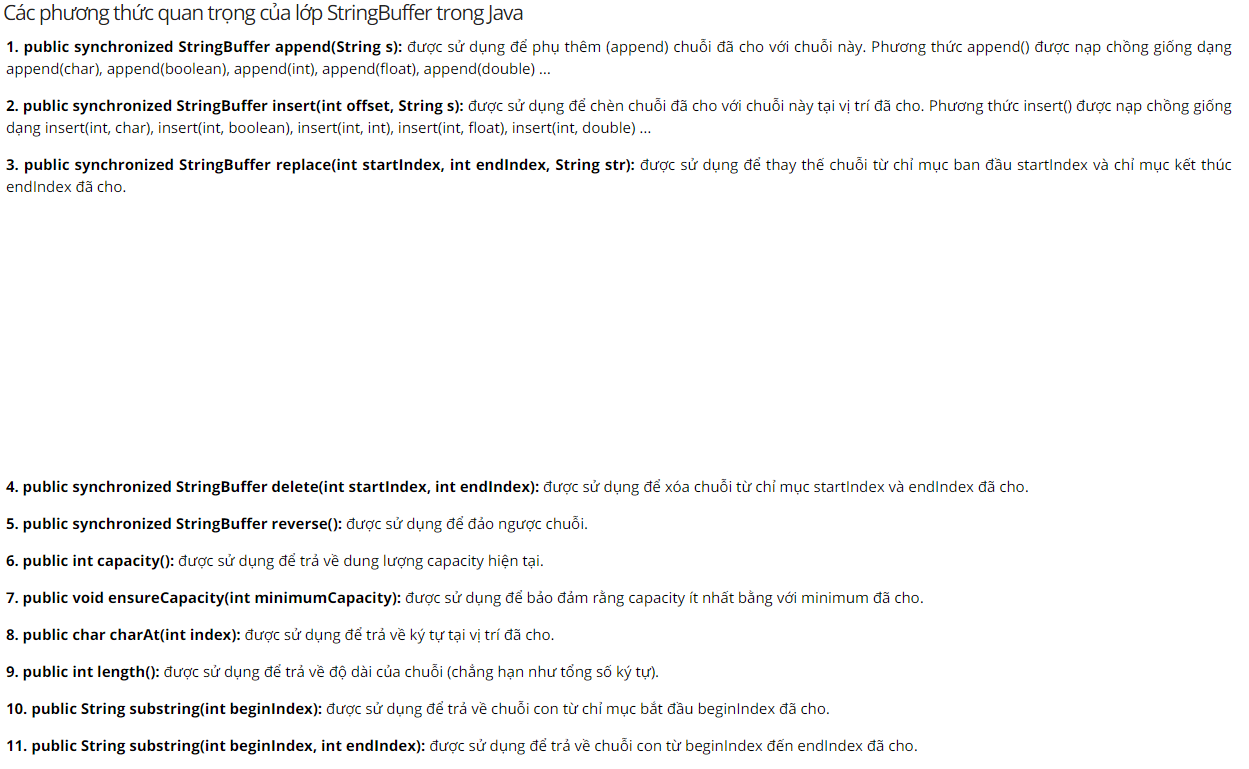
* String : Là đối tượng biểu diễn 1 chuỗi các giá trị char
* StringBuffer được sử dụng để tạo chuỗi có thể thay đổi (mutable)
* StringBuilder được sử dụng để tạo chuỗi có thể thay đổi (mutable) nhưng không đồng bộ (non-synchronized)
* So sánh

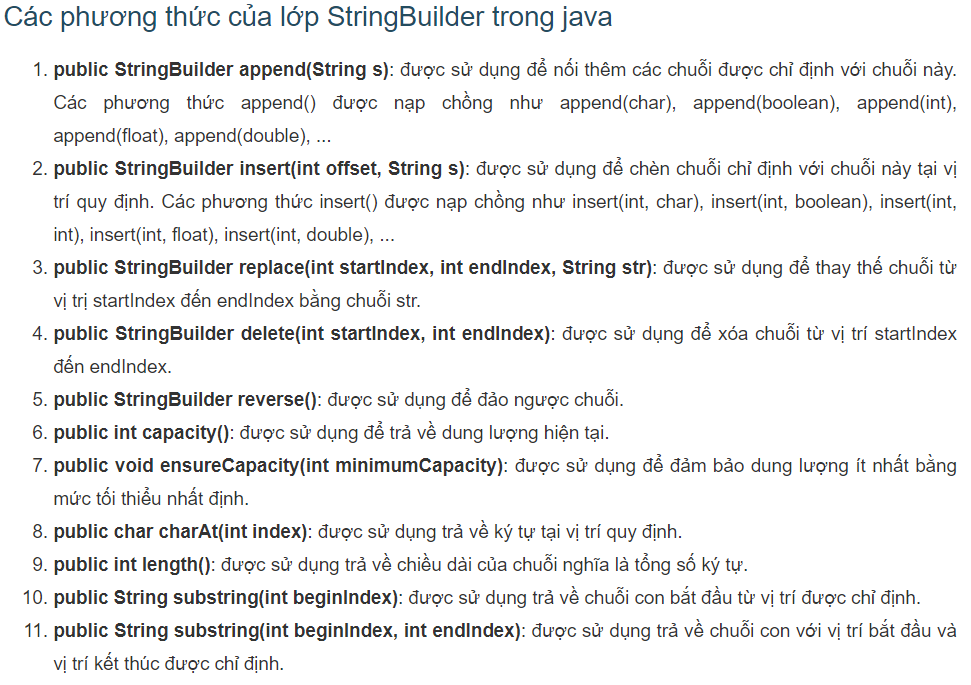
|  |  |
| --- | --- |
| **String** | **StringBuffer** |
| Là lớp bất biến- không thể thay đổi(immuntable) | Là lớp có thể sửa đổi(muntable) |
| Khi nối nhiều chuỗi thì lớp String xử lí chậm và tốn nhiều bộ nhớ vì mỗi lần nối nó lại tạo ra instance mới | Thực hiện nỗi nhiều chuỗi StringBuffer xử lí nhanh và tốn ít bộ nhớ hơn String |
| Lớp String ghi đè phương thức equal() của lớp Object.Vì thế có thể so sánh nội dung 2 chuỗi bằng lớp equal(). | Không thể ghi đè phương thức equal() của lớp Object |

|  |  |
| --- | --- |
| **StringBuffer** | **StringBuilder** |
| Là đồng bộ (synchronized) twsc là luồng an toàn. Có nghĩa là k thể có 2 luồng truy cập phương thức StringBuffer đồng thời | Là không đồng bộ(non-synchronized) tức là luồng k an toàn . Điều này có nghĩa là có 2 luồng cùng truy cập phương thức của lớp StringBuilder đồng thời |
| Kém hiệu quả hơn StringBuilder | Hiệu quả hơn StringBuffer |

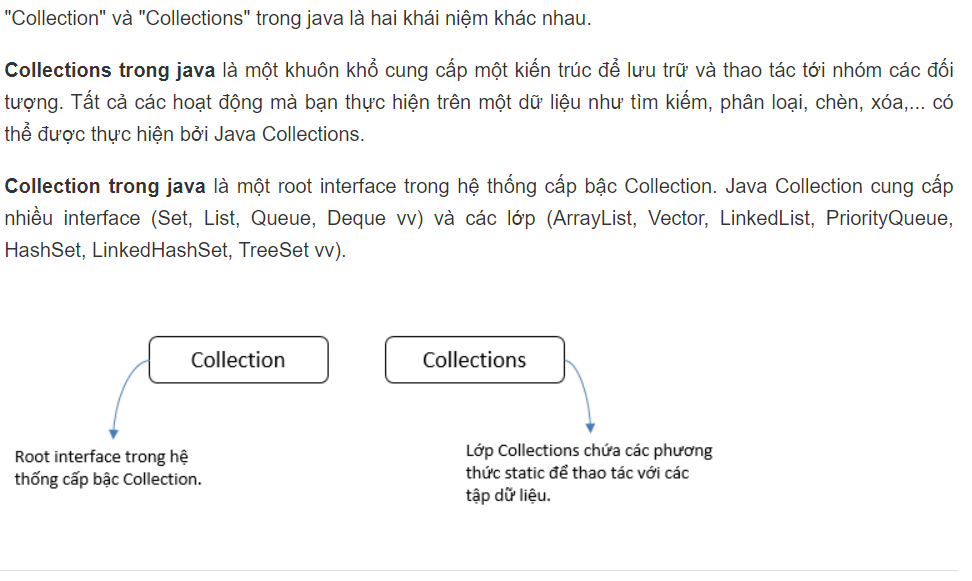


Khi sử dụng Stringbuffer và StringBuilder cần khai báo đối tương mới, sử dụng từ khóa new

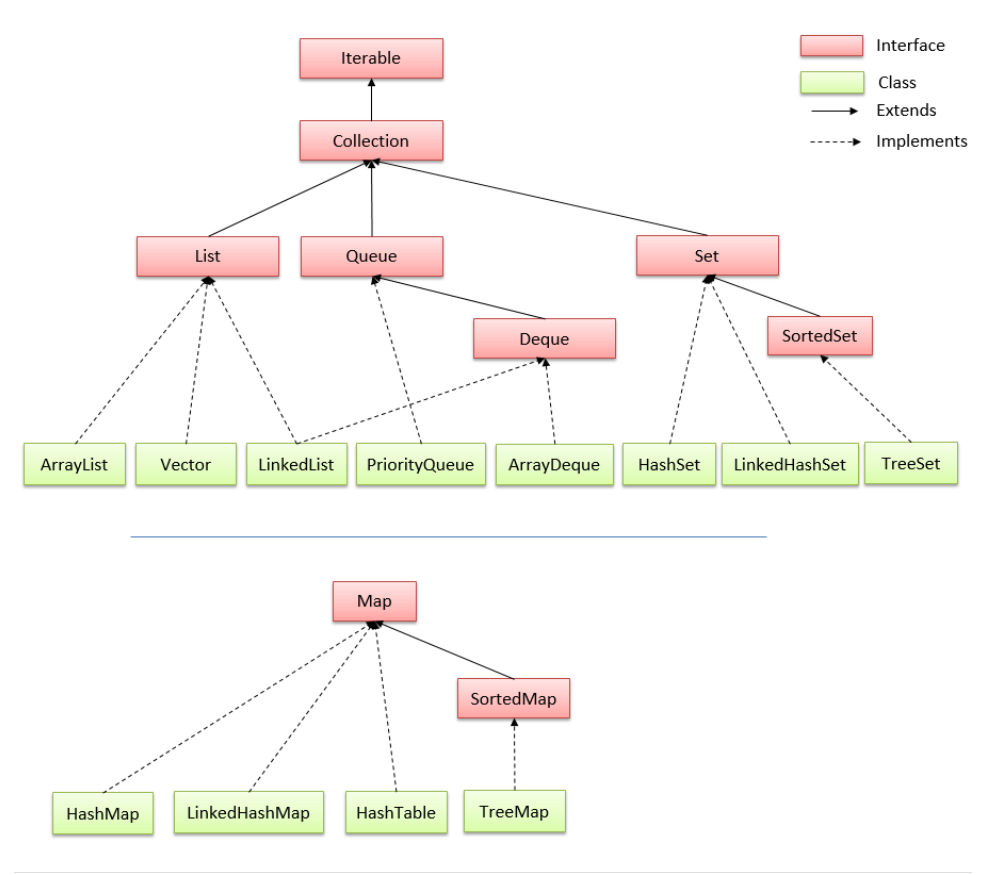




**Câu 9 : Collection trong java**

****

**Hệ thống cấp bậc của collection trong java**

****

* Collections trong java: Là tập hợp các lớp dùng để lưu trữ danh sách và thao tác tới nhóm đối tượng,có khả năng tự co giãn khi thay đổi (như thêm, sửa ,xóa ,chèn,phân loại …) có thể thực hiện bởi Java collections
* Collection trong java là 1 root interface trong hệ thống cấp bậc Collection.Java Collection cung cấp nhiều interface (Set,list,map,…) và các lớp (Araylist,LinkedList,HashSet,LinkedHashSet,TreeSet ,…)
* Iterator là phương thức giao tiếp , cung cấp phương tiện lặp đi lặp lại các thành phần từ đầu tới cuối của 1 collection.
* Iterable interface chứa dữ liệu thành viên của iterator interface
* Ngoài ra còn để lưu trữ truy xuất dữ liệu giữa phương thức với nhau.
* Các interface collections :

1: List Interface :

- LinkedList : Phần tử trong list được sắp xếp có thứ tự và có thể có giá trị giống nhau.

- ArrayList : Thứ tự sắp xếp theo thứ tự nhập vapf và có thể có giá trị giống nhau 2: Set Interface

- Các phần tử trong set là duy nhất (p.tử k có giá trị giống nhau)

- SortedSet : là dạng riêng của set, giá trị phần tử đc sắp xếp tăng dần

- HashSet : k dựa theo thứ tự lúc thêm vào

- TreeSet : các phần tử được sắp xếp tăng dần

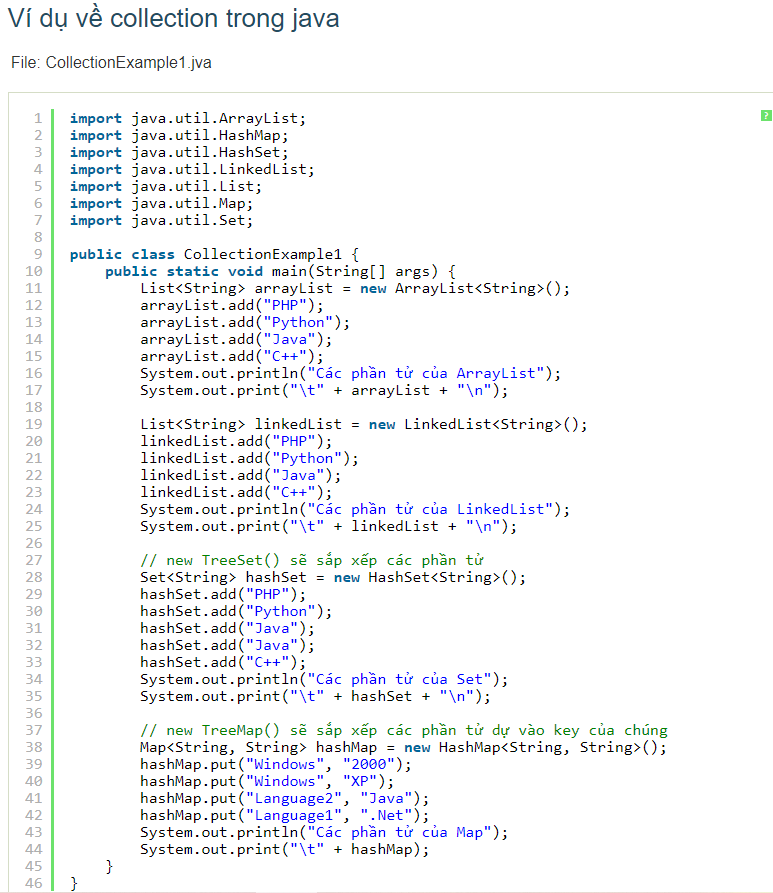
3: Map Interface

- 1 P.tử trong map bao gôm (Key , value ) . Các key trong map k đc trùng nhau

- Tương tự như sortedSet , sortedMap sẽ có key tăng dần

- HashMap : thứ tự phần tử k đực dựa theo thứ tự thêm vào

- TreeMap : thứ tự phần tử được xếp tăng dần dựa theo Key



**Câu 10: Generic**

- 1 lớp có thể tham chiếu bất kì đối tượng nào gọi là generic

- quy ước đặt tham số : (quy ước cho dễ nhớ chứ đặt sao cũng được)

T = Type

E = Element

K = Key

N = Number

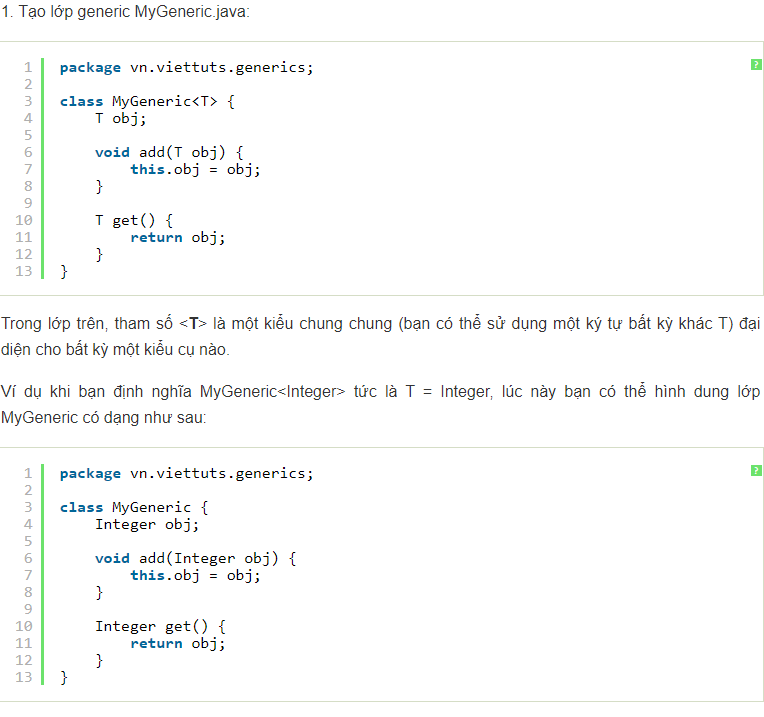
V = Value

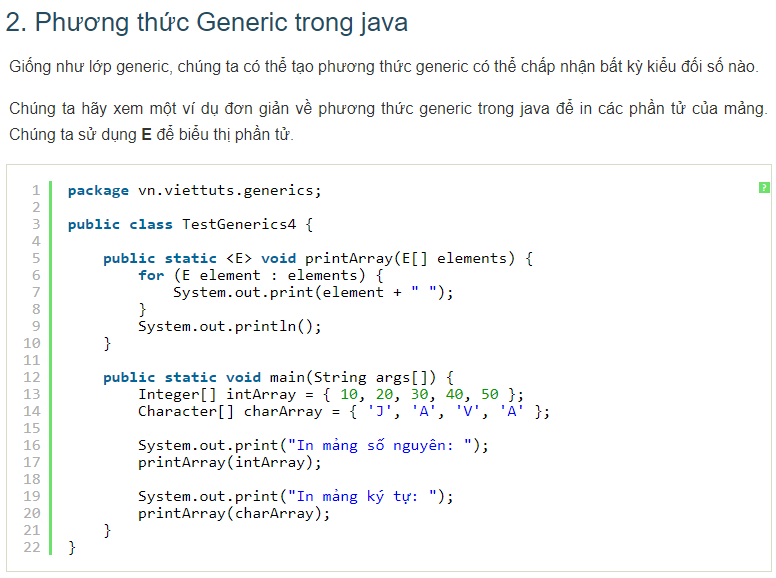
+ lợi thế :

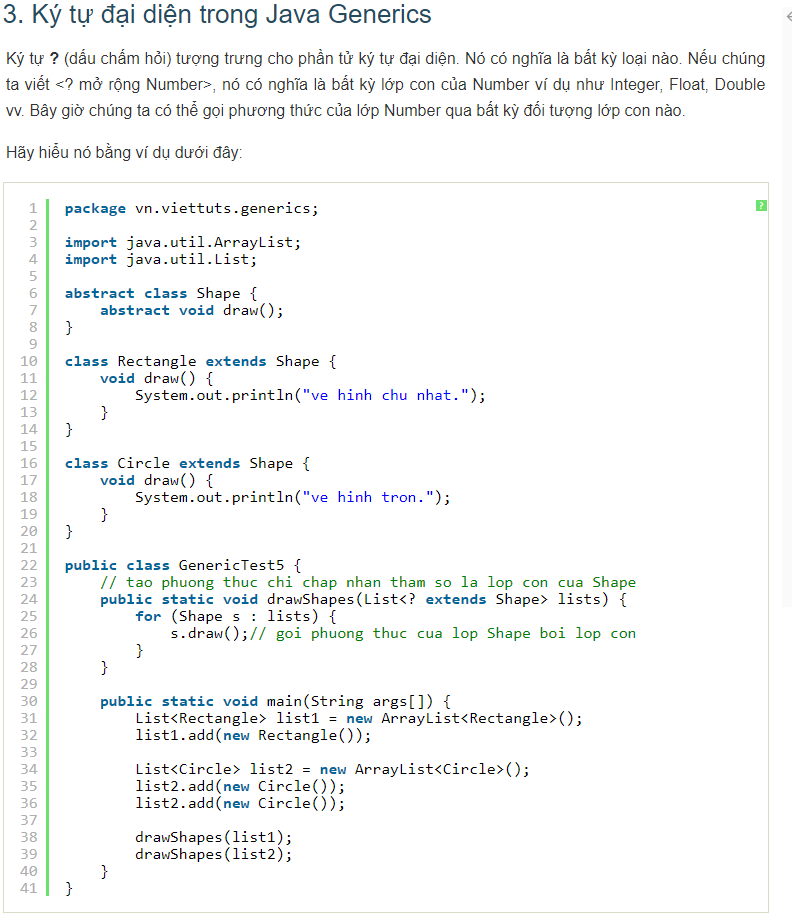
- Chỉ lưu được 1 kiểu đối tượng duy nhất trong generic

- Không cần ép kiểu

- Kiểm tra lúc biên dịch , k bị lỗi khi run time.







**Câu 11: Hàm Equal() , toString,Hashcode**

1: toString : là hàm biểu diễn đối tượng trong Java .Phương thức này trả về biểu diễn chuỗi đối tượng

2: equal là hàm để so sánh 1 chuỗi ,1 đối tượng với 1 đối tượng đã cho. Nếu kết quả là True thì tham số k null và là một đối tượng String để biểu diễn cùng dãy kí tự như đối tượng này.

3: hashCode : gọi hàm hashCode nó sẽ trả vê 1 chuỗi kí tự( địa chỉ ).

+ So sánh :equal với hashCode :

- nếu 1 đối tượng bằng nhau thì có cùng hashCode

- nếu 2 đối tượng có cùng hash code thì chúng có thể bằng nhau hoặc không.

+ So sánh : equal với “==” :

|  |  |
| --- | --- |
| **Equal** | **==** |
| Định nghĩa thế nào là 2 đối tượng trùng nhau, chỉ áp dụng cho đối tượng k áp dụng cho kiểu nguyên thủy | Dùng so sánh địa chỉ 2 đối tượng và giá trị của các biến nguyên thủy |

**Câu 12: 4 tính chất của lập trình hướng đối tượng trong java (OOP in java)**

1: Tính đóng gói (encapsulation), che giấu thông tin

+ Trạng thái của đối tượng được bảo vệ , không cho phép các truy cập từ code bên ngoài vào.

Việc cho phép bên ngoài tác động lên đối tượng trong code phụ thuộc hoàn toàn vào người viết mã.

Đảm bảo toàn vẹn đối tượng và bảo mật trong java thể hiện thông qua phạm vi truy cập.

Ngoài ra các lớp liên quan đến nhau có thể được gom chung lại thành 1 pakage.

2:Tính kế thừa (Inheritance)

+ Là khả năng xây dựng 1 lớp Con mới dựa trên lớp Cha đã có.

Lớp con có thể chia sẻ hay mở rộng những đặc tính đã có mà không phải tiến hành định nghĩa lại

3:Tính đa hình (polymorphism)

+ khi 1 tác vụ thực hiện được theo nhiều cách khác nhau gọi là tính đa hình.

+ Nó được thể hiện rõ nhất khi gọi phương thức của đối tượng.

Các phương thức giống nhau nhưng việc xử lí luồng có thể khác nhau

+Trong java có 2 phương thức sử dụng có tính đa hình

**- OverLoading**: Còn gọi là nạp chồng , cho phép 1 lớp có nhiều thuộc tính , phương thức cùng tên

nhưng với các tham số khác nhau về loại và số lượng. Khi được gọi , dựa vào tham số truyền vào, phương thức tương ứng sẽ được thực hiện.

**- OverRiding:**Cùng tên , tham số và kiểu trả về nhưng thằng con viết lại và dùng theo cách của nó.Lúc thực thi lớp con k có phương thức riêng

thì phương thức thằng cha sẽ được gọi. ngược lại nếu có thì phương thức của thằng con sẽ được gọi.

4: Trừu tượng (Abstraction)

+Ẩn chi tiết triển khai , chỉ hiển thị tính năng tới người dùng.

+Loại bỏ tính chất phức tạp và chỉ đưa ra các thuộc tính, phương thức cần của đối tượng trong lập trình.

+ Giúp người làm tập chung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm đến cách nó thực hiện

+ Trong java sử dụng abstract class và abstract interface để có tính trừu tượng.

**Câu 13:Phân biệt final và finally**

* Final là 1 từ khóa ----- finally là 1 khối (block)
* Final : là từ dùng để áp dụng hạn chế về class , phương thức , biến .Lớp final không thể được kế thừa, phương thức k thể ghi đè và giá trị biến k thể thay đổi
* Finally : Luôn được dùng để thực thi code quan trọng. Nó luôn được thực hiện dù ngoại lệ có được xử lí hay k

**Câu 14 Phân biệt :**

* ArrayList và HashMap

|  |  |  |
| --- | --- | --- |
|  | Khác | |
| Giống | ArrayList | HashMap |
| Đều không được đồng bộ hóa, không nên sử dụng trong môi trường đa luồng | Thực hiện giao diện ở dạng danh sách | Thực hiện giao diện ở dạng bản đồ |
| Iterator thực hiện không nhanh, sẽ ném ra exception khi thay đổi cấu trúc sau khi iterator được tạo | Tiêu thụ bộ nhớ thấp hơn HashMap | Tiêu thụ bộ nhớ cao hơn |
| Có thể null | Có thứ tự trả về | Không có thứ tự |
| Cho phép các giá trị phần tử trùng lặp | Các phần tử trùng lặp | Các giá trị được trùng nhưng khóa thì không |
| Cung cấp hiệu xuất thời gian không đổi nếu sử dụng hàm get() | Cho phép cho 1 hoặc nhiều phần tử được null | Chỉ 1 khóa được null và bất kì giá trị số null nào |
| Được hỗ trợ bởi mảng tương tự.  Đều được duyệt qua iterator | Phương thức get():  Lấy phần tử bằng cách chỉ định mục | Phương thức get():  Tìm bằng khóa tương ứng |

* Array và ArrayList:

|  |  |
| --- | --- |
| Array | ArrayList |
| Kích thước cố định | Kích thước có thể thay đổi |
| Lưu dữ liệu kiểu nguyên thủy và đối tượng | Lưu dữ liệu kiểu đối tượng. Kiểu nguyên thủy được tự động chuyển đổi sang đối tượng (auto-boxing) |
| Tốc độ lưu trữ và thao tác nhanh hơn | Tốc độ lưu trữ và thao tác chậm hơn |
| Chỉ có thuộc tính lenght | Có nhiều phương thức để thao tác với dữ liệu |

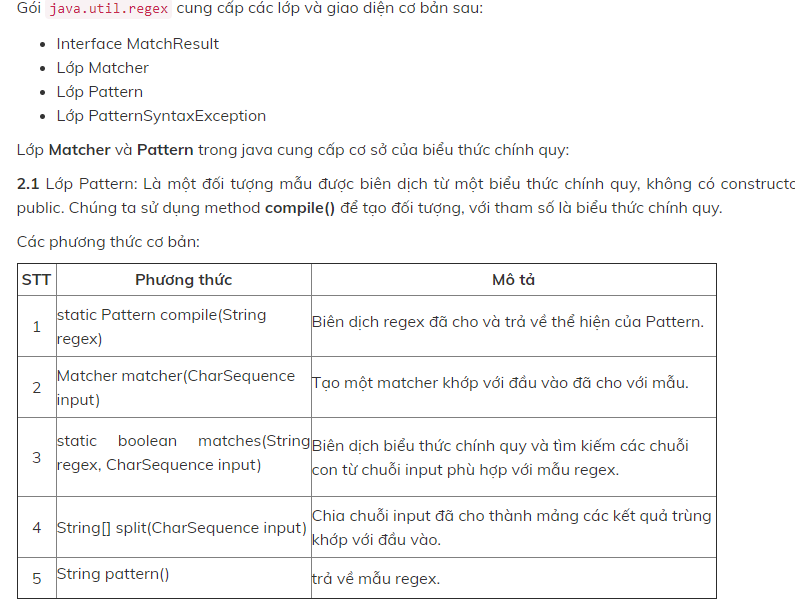
* ArrayList và LinkedList:

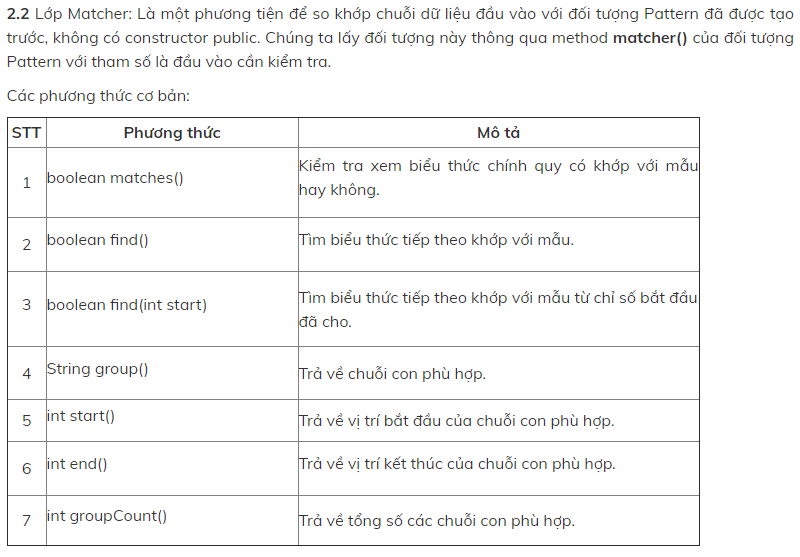
|  |  |
| --- | --- |
| ArrayList | LinkedList |
| Nội bộ sử dụng mảng động để lưu trữ các phần tử | Nội bộ sử dụng danh sách liên kết đôi (boubly) để lưu trữ phần tử |
| Thao tác với ArrayList chậm vì sử dụng nội bộ mảng. Nếu bất kì phần tử nào được xóa khỏi mảng, tất cả các bit được chuyển trong bộ nhớ | Nhanh hơn so với ArrayList vì sử dụng danh sách liên kết đôi, do đó không cần chuyển đổi bit vào trong bộ nhớ |
| Chỉ có thể hoạt động như 1 list vì chỉ implements giao tiếp List | Hoạt động như 1 list và queue(hàng đợi) vì nó implement các giao tiếp list và Deque |
| Tốt hơn trong việc lưu trữ và truy cập dữ liệu | Tốt hơn trong việc thao tác dữ liệu |

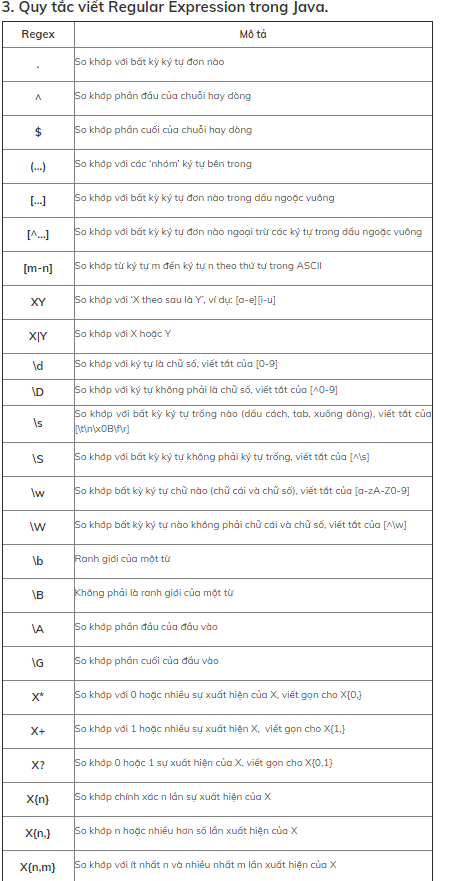
**Câu 15: Regular Expression**

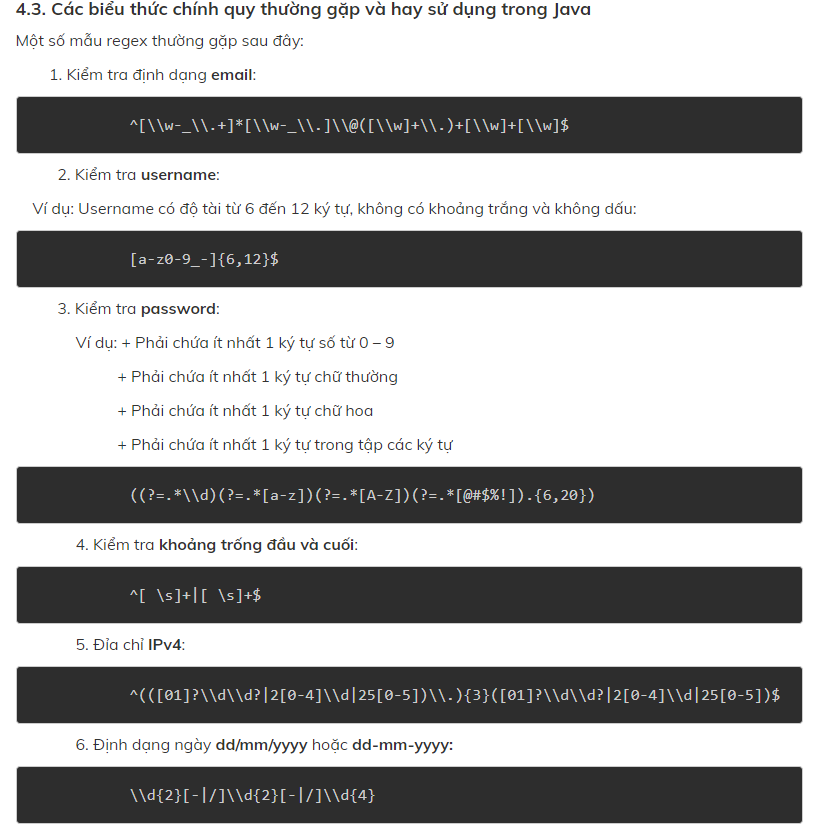
**KN**: Là một dãy liên tục của các kí tự đặc biệt giúp bạn so khớp , tìm kiếm chuỗi hoặc tập hợp các chuỗi,

sử dụng 1 cú pháp riêng biệt trong pattern. Dùng để tìm chỉnh sửa và thao tác dữ liệu text.

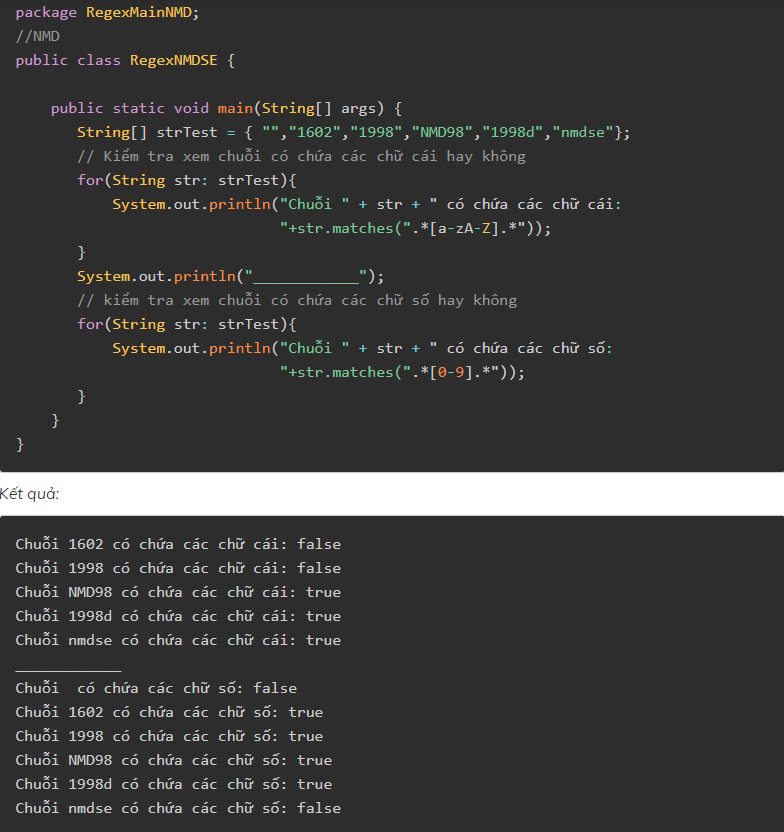






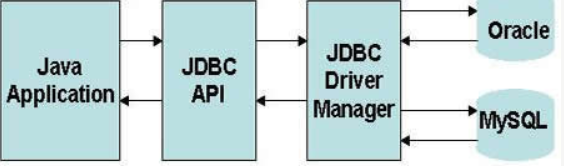


Ví Dụ:



***Ngoài khái niệm ra thì còn lại k hiểu***

**Câu 16: Hiểu biết về JDBC ?**

****

* **KN:** JDBC là viết tắt của Java Database Connectivity. Là một API tiêu chuẩn dùng để tương tác với các loại cơ sở dữ liệu quan hệ . JDBC có 1 tập hợp các class và các Interface dùng cho ứng dụng Java có thể nói chuyện với các cơ sở dữ liệu
* **Thành phần:**

1. DriverManage : Là 1 class dùng để quản lí danh sách driver
2. Driver: Là 1 Interface dùng để liên kết các liên lạc với cơ sở dữ liệu, điều khiên các liên lạc với database.Một khi driver được tải lên, lập trình viên không cần phải gọi 1 cách cụ thể
3. Connection: Là 1 interface với tất cả các method cho việc liên lạc với database. Nó mô tả nội dung liên lạc , tất cả thông tin liên lạc với CSDL , là thông qua chỉ có đối tượng Connection.
4. Statament:Là 1 interface gọi gọn 1 câu lệnh SQL gửi tới CSDL được phân tích, tổng hơp, lập kế hoạch và thực hiện
5. ResuilSet : Là đại diện cho tập hợp các bản ghi lấu do thực hiện truy vấn.

* **Java sử dụng JDBC để làm việc với các CSDL**