viettel
software services

CHƯƠNG TRÌNH ĐÀO TẠO JAVA NÂNG CAO

Java Collection

Đơn vị tổ chức: Phòng đào tạo và phát triển nguồn nhân lực

viettel
software services

NỘI DUNG HỌC PHẦN

BÀI 1

Java Collection

BÀI 2

Java Generic và Reflection

viettel
software services

BÀI 1

Java Collection

viettel
software services

MỤC LỤC

viettel
software services

01

Java Collection Framework

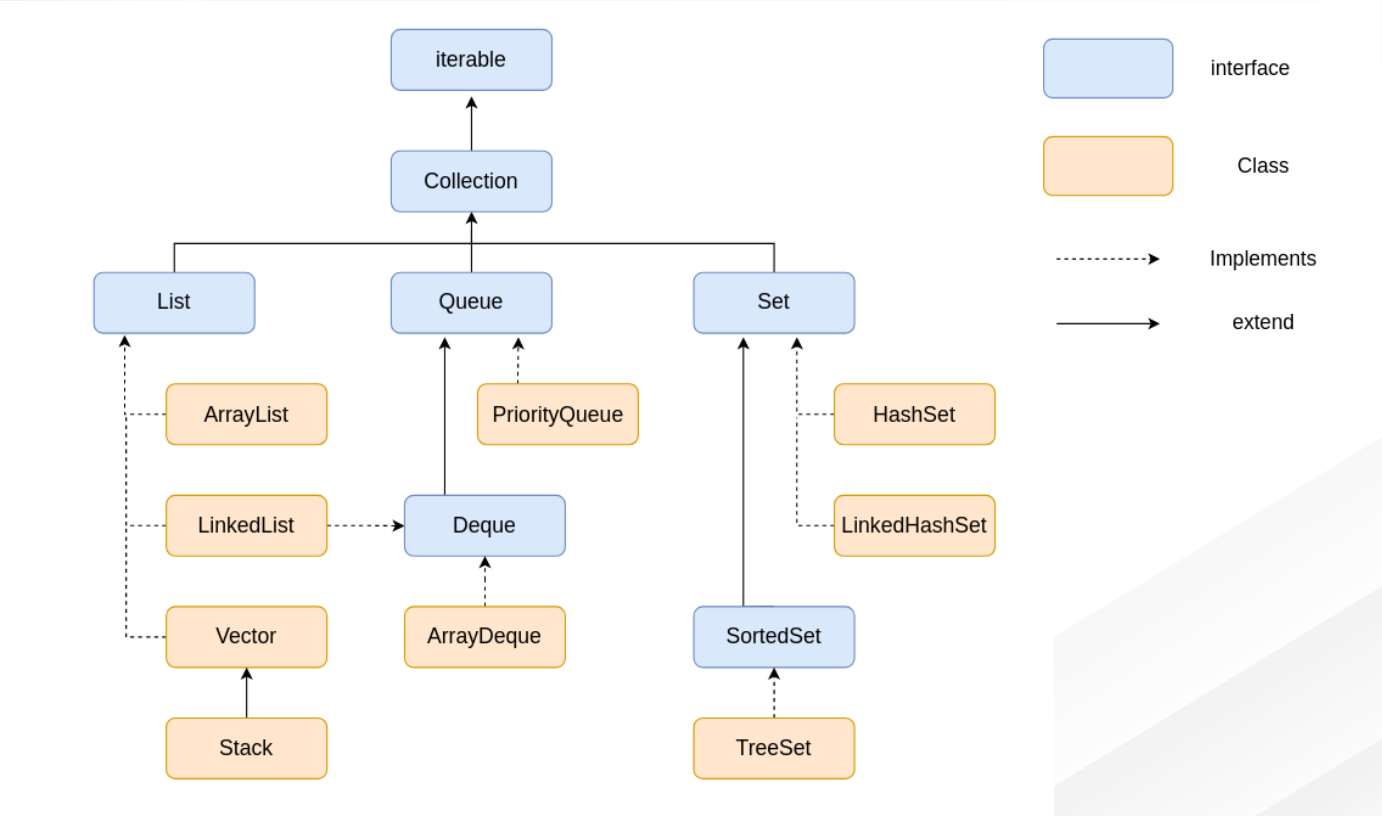
viettel
software services

Khái niệm

- **Java Collection Framework** cung cấp một kiến trúc để lưu trữ và thao tác với một nhóm đối tượng. Một Java Collection Framework bao gồm:
- **Interfaces**: Interface trong Java đề cập đến các kiểu dữ liệu trừu tượng. Chúng cho phép Java collections được thao tác độc lập với các biểu diễn của chúng. Ngoài ra, chúng tạo thành một hệ thống phân cấp trong ngôn ngữ lập trình hướng đối tượng.
- **Classes**: Các lớp trong Java là sự triển khai của collection interface. Nó đề cấp đến các cấu trúc dữ liệu được sử dụng lặp đi lặp lại.
- **Algorithm**: Thuật toán đề cập đến các phương pháp được sử dụng để thực hiện các hoạt động như tìm kiếm và sắp xếp trên đối tượng triển khai của collection interface. Các thật toán có bản chất đa hình vì cùng một phương pháp có thể sử dụng được nhiều dạng hoặc bạn có thể nói có nhiều cách triển khai khác nhau của java collection interface

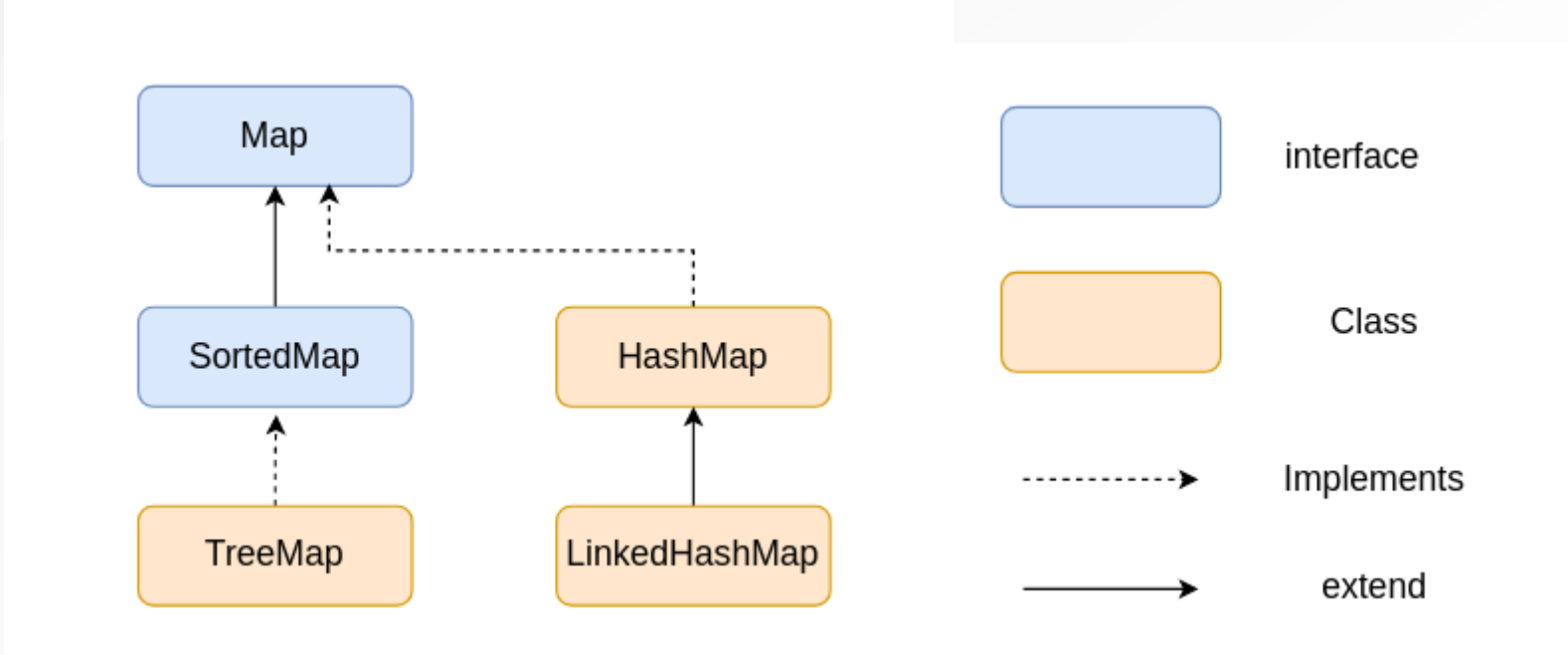
viettel
software services

Cấu trúc phân cấp Java Collection Framework



viettel
software services

Cấu trúc phân cấp Java Collection Framework



Collection Interface

- **Collection Interface** định nghĩa những phương thức cơ bản khi làm việc với tập hợp, đây là gốc cũng là nền móng để từ đó xây dựng lên cả bộ thư viện **Java Collection Framework**. **Collection Interface** được kế thừa từ **Iterable Interface** nên các bạn có thể dễ dàng duyệt qua từng phần tử thông qua việc sử dụng **Iterator**.

Set Interface

- **Set** (tập hợp) là kiểu dữ liệu mà bên trong nó mỗi phần tử chỉ xuất hiện duy nhất một lần (tương tự như tập hợp trong toán học vậy) và **Set Interface** cung cấp các phương thức để tương tác với set. **Set Interface** được kế thừa từ **Collection** Interface nên nó cũng có đầy đủ các phương thức của **Collection Interface**. Một số class thực thi **Set Interface** thường gặp:
- **TreeSet**: là 1 class thực thi giao diện **Set Interface**, trong đó các phần tử trong set đã được sắp xếp.
- **HashSet**: là 1 class implement **Set Interface**, mà các phần tử được lưu trữ dưới dạng bảng băm (hash table).
- **EnumSet**: là 1 class dạng set như 2 class ở trên, tuy nhiên khác với 2 class trên là các phần tử trong set là các enum chứ không phải object.

List Interface

- **List** (danh sách) là cấu trúc dữ liệu tuyến tính trong đó các phần tử được sắp xếp theo một thứ tự xác định. **List Interface** định nghĩa các phương thức để tương tác với list cũng như các phần tử bên trong list. Tương tự như **Set Interface**, **List Interface** cũng được kế thừa và có đầy đủ các phương thức của **Collection Interface**. Một số class thực thi List Interface thường sử dụng:
- **ArrayList**: là 1 class dạng list được implement dựa trên mảng có kích thước thay đổi được.
- **LinkedList**: là một class dạng list hoạt động trên cơ sở của cấu trúc dữ liệu danh sách liên kết đôi (double-linked list)
- **Vector**: là 1 class thực thi giao diện **List Interface**, có cách thực lưu trữ như mảng tuy nhiên có kích thước thay đổi được, khá là tương tự với **ArrayList**, tuy nhiên điểm khác biệt là Vector là synchronized, hay là đồng bộ, có thể hoạt động đa luồng mà không cần gọi synchronize một cách tường minh
- **Stack**: cũng là 1 class dạng list, Stack có cách hoạt động dựa trên cơ sở của cấu trúc dữ liệu ngăn xếp (stack) với kiểu vào ra LIFO (last-in-first-out hay vào sau ra trước) nổi tiếng.

Queue Interface

- **Queue** (hàng đợi) là kiểu dữ liệu nổi tiếng với kiểu vào ra FIFO (first-in-first-out hay vào trước ra trước), tuy nhiên với **Queue Interface** thì queue không chỉ còn dừng lại ở mức đơn giản như vậy mà nó cũng cấp cho bạn các phương thức để xây dựng các queue phức tạp hơn nhiều như priority queue (queue có ưu tiên), deque (queue 2 chiều), … Và cũng giống như 2 interface trước, **Queue Interface** cũng kế thừa và mang đầy đủ phương thức từ **Collection Interface**. Một số class về **Queue** thường sử dụng:
- **LinkedList**: chính là LinkedList mình đã nói ở phần List
- **PriorityQueue**: là 1 dạng queue mà trong đó các phần tử trong queue sẽ được sắp xếp.
- **ArrayDeque**: là 1 dạng deque (queue 2 chiều) được implement dựa trên mảng

viettel
software services

Map Interface

- **Map** (đồ thị/ánh xạ) là kiểu dữ liệu cho phép ta quản lý dữ liệu theo dạng cặp key-value, trong đó key là duy nhất và tương ứng với 1 key là một giá trị value. **Map Interface** cung cấp cho ta các phương thức để tương tác với kiểu dữ liệu như vậy. Không giống như các interface ở trên, Map Interface không kế thừa từ **Collection Interface** mà đây là 1 interface độc lập với các phương thức của riêng mình. Dưới đây là một số class về Map cần chú ý:
- **TreeMap**: là class thực thi giao diện **Map Interface** với dạng cây đỏ đen (Red-Black tree) trong đó các key đã được sắp xếp. Class này cho phép thời gian thêm, sửa, xóa và tìm kiếm 1 phần tử trong Map là tương đương nhau và đều là O(log(n))
- **HashMap**: là class thực thi giao diện **Map Interface** với các key được lưu trữ dưới dạng bảng băm, cho phép tìm kiếm nhanh O(1).
- **EnumMap**: cũng là 1 Map class nữa, tuy nhiên các key trong Map lại là các **enum** chứ không phải object như các dạng Map class ở trên.
- **WeakHashMap**: tương tự như **HashMap** tuy nhiên có 1 điểm khác biệt đáng chú ý là các key trong Map chỉ là các Weak reference (hay Weak key), có nghĩa là khi phần tử sẽ bị xóa khi key được giải phóng hay không còn một biến nào tham chiếu đến key nữa.

viettel
software services

Collections

- **Collections** là một lớp tiện ích. **Collections** bao gồm các phương thức static được sử dụng để thao tác trên các đối tượng của Collection (List, ArrayList, LinkedList, Map, Set, …).
- Colletions có các thuộc tính sau: **static List EMPTY_LIST:** khởi tạo một danh sách trống, không thể thay đổi (immutable). **static Map EMPTY_MAP:** khởi tạo một map trống, không thể thay đổi (immutable). **static Set EMPTY_SET:** khởi tạo một tập hợp trống, không thể thay đổi (immutable).
- Collection cung cấp các hàm xử lý như: sort(), reverse(), min(), max(), list()….

02

Streams

viettel
software services

Streams

- **Stream** (luồng) là một đối tượng mới của Java được giới thiệu từ phiên bản Java 8, giúp cho việc thao tác trên collection và array trở nên dễ dàng và tối ưu hơn.
- Một **Stream** đại diện cho một chuỗi các phần tử hỗ trợ các hoạt động tổng hợp tuần tự (sequential) và song song (parallel).
- Tất cả các class và interface của **Stream API** nằm trong gói **java.util.stream**. Bằng cách sử dụng các stream, chúng ta có thể thực hiện các phép toán tổng hợp khác nhau trên dữ liệu được trả về từ các collection, array, các hoạt động Input/Output.
- Trong Java 8, Collection interface được hỗ trợ 2 phương thức để tạo ra Stream bao gồm: **stream() :** trả về một stream sẽ được xử lý theo tuần tự. **parallelStream() :** trả về một Stream song song, các xử lý sau đó sẽ thực hiện song song.

viettel
software services

Các đặc điểm của Streams

- **Stream** không lưu trữ các phần tử của collection hay array. Nó chỉ thực hiện các phép toán tổng hợp
- **Stream** không phải là một cấu trúc dữ liệu (data structure).
- **Stream** là **immutable object**. Các hoạt động tổng hợp mà chúng ta thực hiện trên Collection, Array hoặc bất kỳ nguồn dữ liệu nào khác không làm thay đổi dữ liệu của nguồn, chúng chỉ trả lại stream mới. Chúng ta đã thấy ở ví dụ trên là thực hiện filter() các các số chẵn bằng cách sử dụng các hoạt động của stream nhưng nó không thay đổi các phần tử của **List numbers**.
- Tất cả các hoạt động stream là **lazy** (lười biếng), có nghĩa là chúng không được thực hiện cho đến khi cần thiết. Để làm được điều này, hầu hết các thao tác với **Stream** đều return lại một **Stream** mới, giúp tạo một mắc xích bao gồm một loạt các thao tác nhằm thực thi các thao tác đó một cách tối ưu nhất. Mắc xích này còn được gọi là pipeline.

Các đặc điểm của Streams

- Các phần tử của luồng chỉ được truy cập một lần trong suốt vòng đời của **Stream**. Giống như một **Iterator**, một **Stream** mới phải được tạo ra để duyệt lại các phần tử của dữ liệu nguồn.
- **Stream** không dùng lại được, nghĩa là một khi đã sử dụng nó xong, chúng ta không thể gọi nó lại để sử dụng lần nữa.
- Chúng ta không thể dùng index để truy xuất các phần tử trong **Stream**.
- **Stream** hỗ trợ thao tác song song các phần tử trong **Collection** hay **Array**.

viettel
software services

So sánh giữa Stream và Colelctions

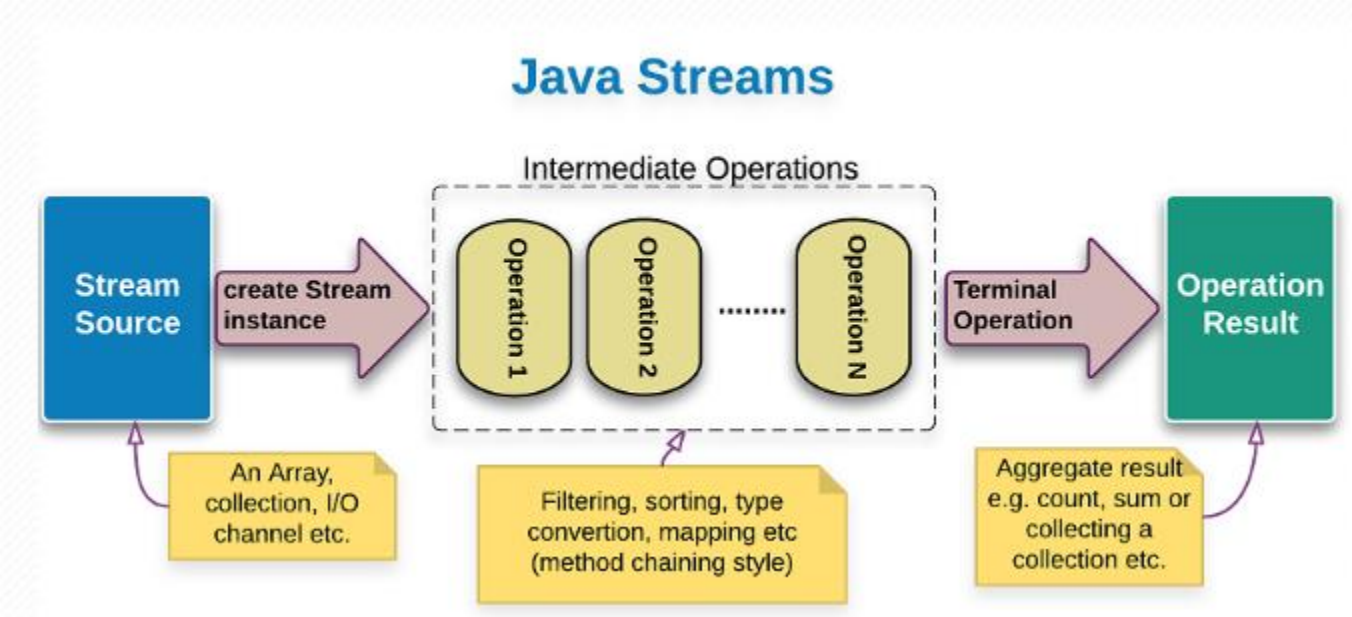
- **Lưu trữ – Java Stream** không giống như các **Collection** không lưu trữ bất kỳ dữ liệu nào.
- **Tính toán** – Các phần tử trong các **Collection** được tính toán trước khi được thêm vào **Collection**, trong khi các phần tử stream được tính theo yêu cầu nếu chúng được sử dụng.
- **Kích thước** – Collection chứa số lượng phần tử hữu hạn, trong khi **Stream** có khả năng vô hạn.
- **Mức tiêu thụ** – Các phần tử trong **Stream** chỉ có thể được sử dụng một lần, tương tự như iterator. Các phần tử trong **Collection** có thể được dùng nhiều lần như mong muốn.
- **Iteration** – Việc lặp lại trên các **collection** được thực hiện bên ngoài (bởi người dùng) trong khi **Stream** sử dụng lặp lại nội bộ và xử lý việc này cho bạn. Do đó việc debug khá khó khăn trên **Streams** do việc lặp nội bộ.

Các Pharse xử lý của Stream

- Stream xử lý các phần tử của tập hợp theo 2 bước:
- **Tạo Stream**
- **Cấu hình - configuration**
- **Xử lý - processing**
- Cấu hình gồm một tác vụ hoặc cả hai là filter và map. Trong quá trình cấu hình thì không có phần tử tập hợp nào thực sự bị tác động, chúng chỉ thực sự được xử lý khi quá trình xử lý thực hiện, tức là phương thức xử lý được gọi.

viettel
software services

Các Pharse xử lý của Stream



viettel
software services

Tạo Stream

- Trong Java 8, **Collection interface** được hỗ trợ 2 phương thức để tạo ra **Stream** bao gồm: **stream()** và **parallelStream()**
- **stream() :** trả về một stream sẽ được xử lý theo tuần tự.
- **parallelStream() :** trả về một Stream song song, các xử lý sau đó sẽ thực hiện song song.

viettel
software services

Các phương thức cấu hình

- **Stream.filter()**: Chúng ta lọc các phần tử của tập hợp bằng cách gọi phương thức **filter()** của **Stream**. Khi phương thức **filter()** được gọi thì chỉ có tham số của bộ lọc được khởi tạo, còn việc lọc (filter) chưc được thực hiện, tức là danh sách trả về nếu có chưa được tạo ra.
- **Stream.map():** Là phương thức mạnh hỗ trợ việc map từng phần tử của danh sách với đối tượng khác. Cũng giống **filter**, lúc này chỉ có việc cấu hình cho việc **mapping** được được hiện chứ việc map chưa diễn ra chừng nào phương thức xử lý chưa được gọi.
- **Stream.skip()** được sử dụng để loại bỏ các phần tử n đầu tiên của Stream . Nếu Stream này chứa ít hơn n phần tử thì luồng trống sẽ được trả lại.
- **Stream.limit()** được sử dụng để cắt giảm kích thước của **Stream**. Kết quả trả về các phần tử của Stream đã được cắt giảm để không vượt quá **maxSize** (tham số đầu vào của phương thức).

Các phương thức xử lý

- **Stream.forEach():** dùng để duyệt các phần tử của Stream
- **Stream.collect():** Phương thức collect() làm một trong những phương thức xử lý tiêu biểu của **interface Stream**. Khi phương thức này được gọi thì việc **filter** hay **mapping** ở trên mới thực sự được thực hiện.
- **Stream.count():** Phương thức **count()** là phương thức xử lý của **Stream**. Nó trả về số lượng phần tử khi thực hiện **filter** của **stream collection**.
- **Stream.reduce():** Phương thức **reduce()** là phương thức xử lý nó giúp làm giảm các phần tử của **stream** về một giá trị đơn lẻ.

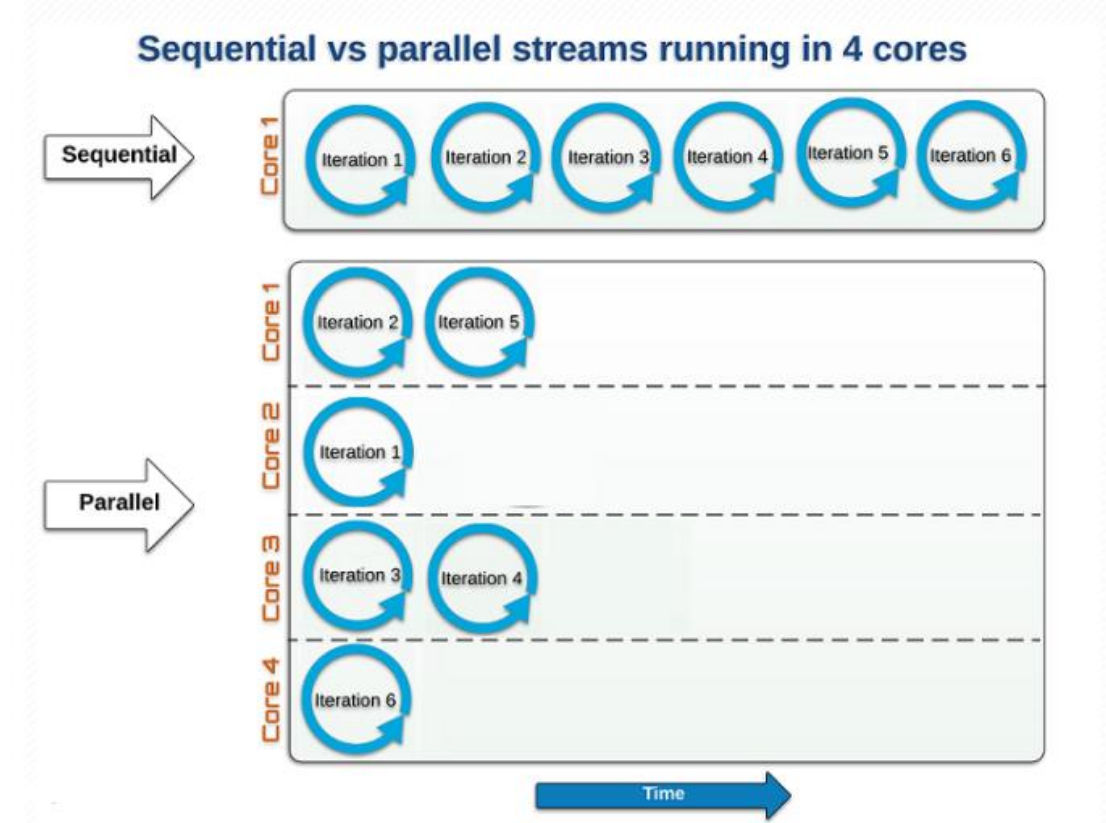
viettel
software services

Cách hoạt động của ParallelStream

- Như đã đề cập ở trên, các stream có thể là **tuần tự (sequential)** hoặc **song song (parallel)**. Các thao tác trên các stream tuần tự được thực hiện trên một **luồng đơn (single thread)** trong khi các phép toán trên các **stream song song** được thực hiện đồng thời trên **nhiều luồng (multi-thread).**
- **Cách sử dụng tương tự với stream()**

viettel
software services

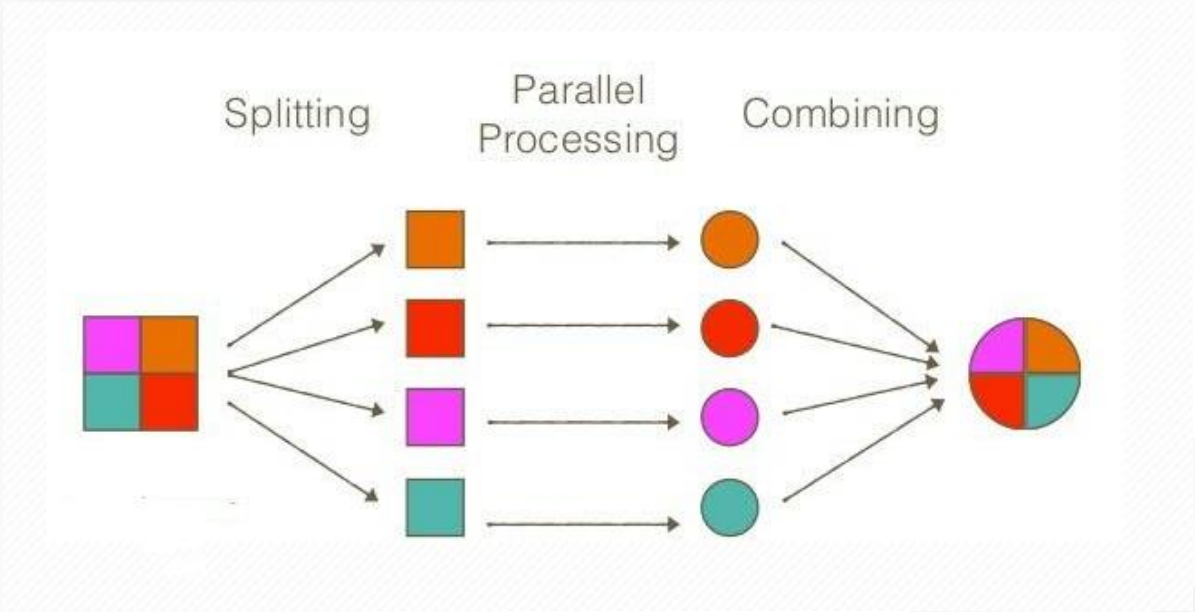
ParallelStream



viettel
software services

Các hoạt động của ParallelStream

- Chúng ta thường sử dụng Parallel Streams trong môi trường multi-thread khi mà chúng ta cần hiệu suất xử lý nhanh.



03

Ví dụ

viettel
software services

Serializable

- **Serializable** trong Java hay tuần tự hóa trong Java là một cơ chế giúp lưu trữ và chuyển đổi trạng thái của 1 đối tượng (Object) vào 1 byte stream sao cho byte stream này có thể chuyển đổi ngược trở lại thành một Object.
- Quá trình chuyển đổi byte stream trở thành 1 Object được gọi là **DeSerialization**.
- Để một Object có thể thực hiện **Serialization** hay gọi tắt là **Serializable**, class của Object cần phải thực hiện implements interface **java.io.Serializable**.
- **java.io.Serializable** là một Interface (giao diện) đánh dấu không có các dữ liệu và phương thức.
- Kỹ thuật của **Serialization** trong Java thường được sử dụng chủ yếu trong công nghệ như RMI, EJB, JPA và Hibernate

viettel
software services

BÀI 2

Java Generic và Reflection

viettel
software services

MỤC LỤC

viettel
software services

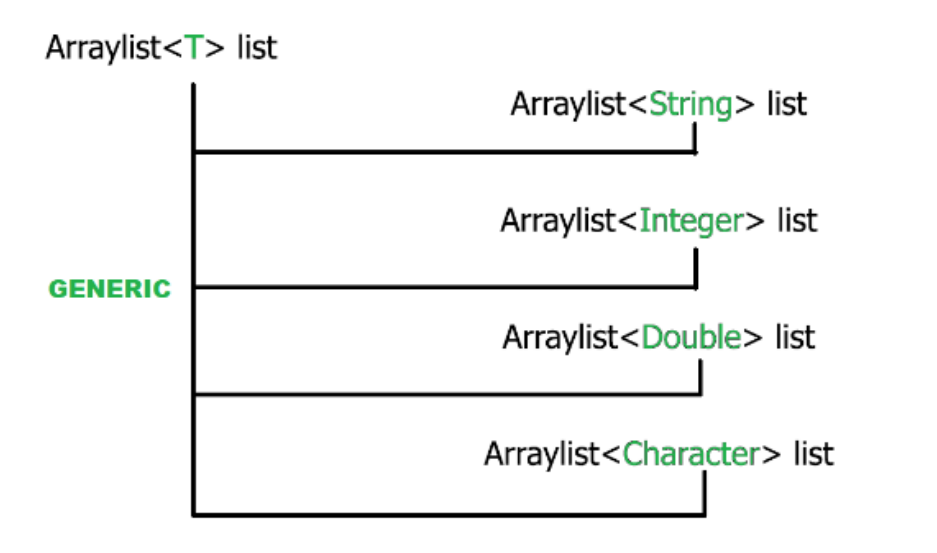
01

Java Generic

viettel
software services

Java Generic

- **Thuật ngữ "generics"** được hiểu là **tham số hóa kiểu dữ liệu**. Việc tham số hóa kiểu dữ liệu giúp cho lập trình viên có thể dễ bắt lỗi các kiểu dữ liệu không hợp lệ, đồng thời giúp dễ dàng hơn cho việc tạo và sử dụng các **class, interface, method** với nhiều kiểu dữ liệu khác nhau.



Một số quy ước trong Generics

- Có rất nhiều cách để đặt tên cho kiểu tham số trong Generic nhưng chúng ta nên tuân theo nhưng kiểu đặt tên tiêu chuẩn để sau này nếu có làm việc nhóm thì team sẽ dễ đọc code hơn.
- Các kiểu tham số thông thường:
- **T - Type** (Kiểu dữ liệu bất kỳ thuộc Wrapper class: String, Integer, Long, Float, …)
- **E – Element** (phần tử – được sử dụng phổ biến trong Collection Framework)
- **K – Key** (khóa)
- **V – Value** (giá trị)
- **N – Number** (kiểu số: Integer, Double, Float, …)
- U,S,I,G, … (tùy theo kiểu của người dùng đặt)

viettel
software services

Ký tự Diamond <>

- Trong Java 7 và các phiên bản sau, bạn có thể thay thế các đối số kiểu dữ liệu cần thiết để gọi hàm khởi tạo (constructor) của một lớp Generic bằng cặp dấu <>. Trình biên dịch sẽ xác định hoặc suy ra các kiểu dữ liệu từ ngữ cảnh sử dụng

```
class QueueOfStrings {
   private LinkedList<String> items = new LinkedList<String>();
   public void enqueue(String item) {
      items.addLast(item);
   }
   public String dequeue() {
      return items.removeFirst();
   }
   public boolean isEmpty() {
      return (items.size() == 0);
   }
}
```

```
class Queue<T> {
   private LinkedList<T> items = new LinkedList<T>();
   public void enqueue(T item) {
      items.addLast(item);
   }
   public T dequeue() {
      return items.removeFirst();
   }
   public boolean isEmpty() {
      return (items.size() == 0);
   }
}
```

viettel
software services

Ký tự đại diện <?>

- Ký tự đại diện <? extends type> chấp nhận bất ký đối tượng nào miễn là đối tượng này kế thừa từ type hoặc đối tượng của type

```
public void processElement(List<? extends A> elements){
 ...
}
```

- Sử dụng phương thức processElement (Chấp nhận bất ký đối tượng nào miễn là đối tượng này phải kế thừa từ lớp A hoặc đối tượng của A => ClassA, ClassB và ClassC)

02

Reflection

viettel
software services

Java Reflection là gì?

- **Java** là một ngôn ngữ hướng đối tượng (Object-oriented), thông thường bạn cần tạo ra một đối tượng và bạn có thể truy cập vào các trường (field), hoặc gọi phương thức (method) của đối tượng này thông qua toán tử dấu chấm (.).
- **Java Reflection** giới thiệu một cách tiếp cận khác, bạn có thể truy cập vào một trường của một đối tượng nếu bạn biết tên của trường đó. Hoặc bạn có thể gọi một phương thức của đối tượng nếu bạn biết tên phương thức, các kiểu tham số của phương thức, và các giá trị tham số để truyền vào …
- **Java Reflecion** cho phép bạn truy cập, sửa đổi cấu trúc và hành vi của một đối tượng tại thời gian chạy (runtime) của chương trình. Đồng thời nó cho phép bạn truy cập vào các thành viên private (private member) tại mọi nơi trong ứng dụng, điều này không được phép với cách tiếp cận truyền thống.
- **Java Reflection** khá mạnh mẽ và rất hữu ích đối với những ai hiểu rõ về nó. Ví dụ, bạn có thể ánh xạ (mapping) đối tượng (object) thành table dưới database tại thời điểm runtime. Kỹ thuật này các bạn có thể thấy rõ nhất ở JPA và Hibernate.

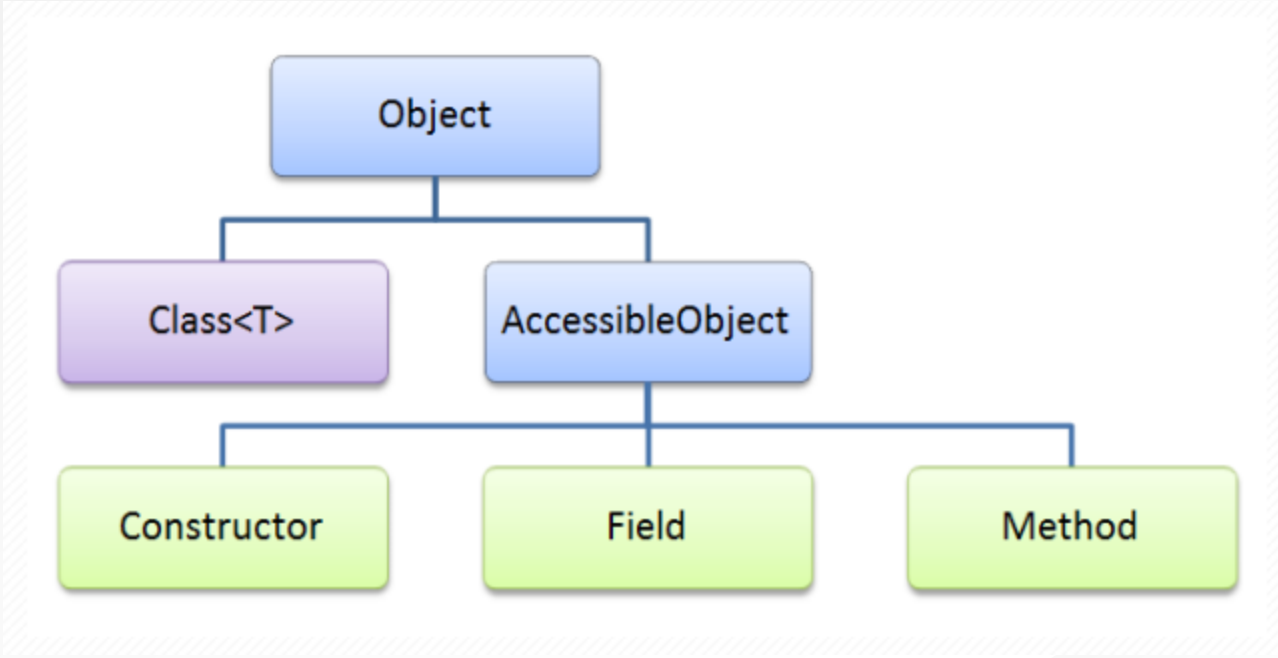
viettel
software services

Kiến trúc của Java Reflection API

- Các lớp được dùng trong **reflection** nằm trong hai package là **java.lang** và **java.lang.reflect**. Package **java.lang.reflect** bao gồm ba lớp chính mà bạn cần biết là **Constructor, Field và Method:**
- **Class<T>:** lớp này đại diện cho các lớp, interface và chứa các phương thức dùng để lấy các đối tượng kiểu **Constructor, Field, Method,…**
- **AccessibleObject**: các kiểm tra về phạm vi truy xuất (public, private, protected) của **field**, **method**, **constructor** sẽ được bỏ qua. Nhờ đó bạn có thể dùng **reflection** để thay đổi, thực thi các thành phần này mà không cần quan tâm đến phạm vi truy xuất của nó.
- **Constructor**: chứa các thông tin về một constructor của lớp.
- **Field**: chứa các thông tin về một field của lớp, interface.
- **Method**: chứa các thông tin về một phương thức của lớp, interface

viettel
software services

Kiến trúc của Java Reflection API



viettel
software services

Lớp (Classes)

- Khi sử dụng **Java Reflection** để duyệt qua một **class** thì việc đầu tiên thường phải làm đó là có được một đối tượng kiểu **Class**, từ các đối tượng kiểu **Class** chúng ta có thể lấy được các thông tin về:
- **Class Name**
- **Class Modifies (public, private, synchronized etc.)**
- **Package Info**
- **Superclass**
- **Implemented Interfaces**
- **Constructors**
- **Methods**
- **Fields**
- **Annotations**

viettel
software services

Tạo đối tượng Class<>

- Đối tượng kiểu Class được tạo ra bằng cách sử dụng phương thức static Class.forName(). Cách này thường được sử dụng khi chỉ biên được tên lớp lúc thực thi (runtime):**Class Name**

```
try {

    Class c =  Class.forName("com.gpcoder.Cat");
    // ...
} catch (ClassNotFoundException e) {
    System.err.println(e);
}
```

- Trong trường hợp không tìm thấy lớp tương ứng, phương thức trên sẽ ném ra ngoại lệ **ClassNotFoundException**. Điều này có thể bất tiện vì bạn phải sử dụng try catch hoặc ném ngoại lệ này khỏi phương thức.

viettel software services

Constructor

- Lấy tất cả Constructor của một Class. Các đối tượng lớp Contructor là những phương thức khởi tạo của một lớp. Reflection cho phép lấy ra những Contructor từ Class Object:

```
Class aClazz = Cat.class; // obtain class object
Constructor[] constructors = aClazz.getConstructors();
```

- Lấy danh sách tham số của một Constructor

```
Constructor constructor = ...; // obtain constructor
Class[] parameterTypes = constructor.getParameterTypes();
```

- Khởi tạo đối tượng từ đối tượng Constructor. Class.newInstance(): tạo một đối tượng với constructor không có tham số. Constructor.newInstance(Object[] initargs): tạo đối tượng với constructor có tham số.

```
Cat cat2 = (Cat) constructor.newInstance("Tom");
```

viettel
software services

Field

- Lấy các đối tượng field được khai báo là public. Bạn có thể lấy được đối tượng field được khai báo là public của một **Class** bằng 2 cách là chỉ lấy một field duy nhất nếu bạn biết chính xác tên của 1 field, hoặc lấy nguyên 1 mảng danh sách các field của từ một đối tượng **Class**.

```
Class aClazz = Cat.class;
Field field = aClazz.getField("name"); // Tên field cần lấy

// Lấy danh sách tất cả các field được khai báo là public
Field[] fields = aClazz.getFields();
```

- Lấy các đối tượng field khai báo bất kỳ. Phương thức **getField()** và **getFields()** chỉ có thể lấy các **field** được khai báo là **public**. Vậy làm sao để access được những **field** được khai báo là **private**, **protected**,…? Khá đơn giản, trong Java có thể lấy được chúng thông qua 2 **methods** là **getDeclaredField()** và **getDeclaredFields().**

viettel software services

Method

- Phương thức Class.getMethods(): trả về danh sách đối tượng Method của một lớp.

```
Class aClazz = Cat.class; // obtain class object
Method[] methods = aClazz.getMethods();
```

- Phương thức **Class.getMethod(String name, Class[] parameterTypes):** trả về đối tượng **Method** đại diện cho một phương thức của lớp. Phương thức này được xác định qua tên và các kiểu tham số.

```
Class aClazz = Cat.class; // obtain class object
Method method = aClazz.getMethod("setName", String.class);
```

- Phương thức **Method.invoke(Object obj, Object[] args)** thực thi phương thức tương ứng của đối tượng **obj** với các tham số **args**.
- Lấy các đối tượng method khai báo bất kỳ sử dụng phương thức **getDeclareMethod()** và **get Declare Fields()**

03

Ví dụ

viettel
software services

XIN CẢM ƠN