



FPT POLYTECHNIC



Conceive Design Implement Operate

JAVA 8 AND JACKSON

GIẢNG VIÊN: NGUYỄN NGHIỆM

www.poly.edu.vn



- Lambda



- Stream API

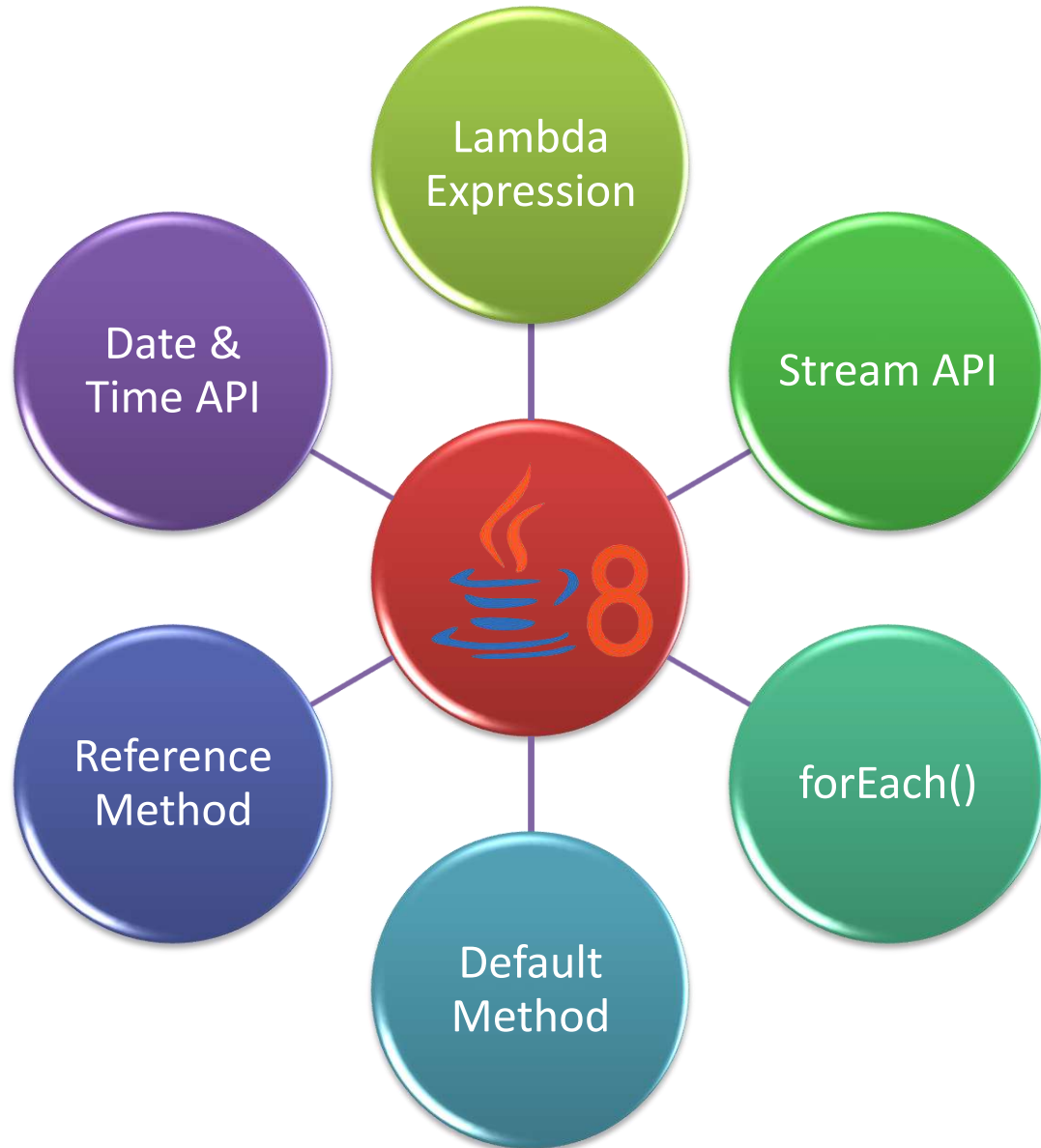


- JSON



- Jackson API





- ☐ Lambda Expression
 - ❖ Kỹ thuật lập trình mới, đơn giản
- ☐ Stream API
 - ❖ Tăng cường xử lý Collection
- ☐ forEach()
 - ❖ Duyệt Collection, Map
- ☐ Default & Static Method
 - ❖ Cho phép viết mã trong interface và đơn giản hóa implements interface
- ☐ Reference Method
 - ❖ Tham chiếu phương thức
- ☐ Date & Time API
 - ❖ Đơn giản hóa xử lý Date và Time

```
List<Integer> list = Arrays.asList(2, 1, 3, 7, 8, 4, 5);  
double a = list.stream()  
    .filter(i -> i % 2 == 0)  
    .peek(System.out::println)  
    .map(n -> Math.sqrt(n))  
    .mapToDouble(d -> d)  
    .average().getAsDouble();  
System.out.println(a);
```



LAMBDA EXPRESSION



```
@Getter @Setter  
@AllArgsConstructor @NoArgsConstructor  
public class Staff {  
    String name;  
    double salary;  
}
```

```
Staff nv1 = new Staff("Tuấn", 100);  
Staff nv2 = new Staff("Hương", 200);  
Staff nv3 = new Staff("Hạnh", 150);  
List<Staff> list = Arrays.asList(nv1, nv2, nv1);
```

```
// TRUYỀN THỐNG  
for(Staff item: list) {  
    System.out.println(item.getName());  
}
```

```
// LAMBDA  
list.forEach(o -> System.out.println(o.getName()));
```

```
Map<String, Staff> map = new HashMap<String, Staff>();  
map.put("NV1", new Staff("Tuấn", 100));  
map.put("NV2", new Staff("Hương", 200));  
map.put("NV3", new Staff("Hạnh", 150));
```

Staff

// TRUYỀN THỐNG

```
for(Entry<String, Staff> entry: map.entrySet()) {  
    String k = entry.getKey();  
    Staff v = entry.getValue();  
    // ...  
}
```

// LAMBDA

```
map.forEach((k, v) -> { /*-...-*/ });
```

List<Staff>

// TRUYỀN THỐNG

```
list.sort(new Comparator<Staff>() {  
    @Override  
    public int compare(Staff o1, Staff o2) {  
        return o1.getName().compareTo(o2.getName());  
    }  
});
```

// LAMBDA

```
list.sort((o1, o2) -> o1.getName().compareTo(o2.getName()));
```


// TRUYỀN THỐNG

```
new Thread(new Runnable() {  
    @Override  
    public void run() { /*-code-*/ }  
}).start();
```

// LAMBDA

```
new Thread(() -> { /*-code-*/ }).start();
```

❑ `Collection<T>.forEach(Consumer<T>)`

❖ `Consumer<T>.accept(T)`

❑ `Map(K, V).forEach(BiConsumer<K, V>)`

❖ `BiConsumer<K, V>.accept(K, V)`

❑ `List<T>.sort(Comparator<T>)`

❖ `Comparator<T>.compare(T, T)`

❑ `Thread(Runnable)`

❖ `Runnable.run()`

Nhận xét: *tham số của các phương thức `forEach()`, `sort()`, `new Thread()` là các **interface chỉ chứa duy nhất 1 phương thức trừu tượng**. Vì vậy khi gọi các phương thức trên có thể truyền biểu thức lambda thay cho đối tượng được tạo ra từ các interface đó.*

([parameters]) -> {method body}

- ❑ A Java lambda expression is an **anonymous method** that can be created without belonging to any class. Instead, it is used **to implement a method defined by a function interface** (*this interface contains one — and only one — abstract method but also can contain multiple default and static methods*)
- ❑ Biểu thức Lambda là phương thức nặc danh nhằm **hiện thực mã nguồn phương thức của @FunctionalInterface** (chỉ khai báo **duy nhất một phương thức trừu tượng**)

FUNCTIONAL INTERFACE

```
@FunctionalInterface
public interface MyFuncInter{
    T0 m1(T1 a, T2 b);
    default Y m2(){...}
    static Z m3(){...}
}
```

@FunctionalInterface chỉ để đảm bảo *interface chỉ chứa một phương thức trừu tượng duy nhất*, ngược lại sẽ nhận được thông báo lỗi tại thời điểm dịch

TRADITIONAL

```
MyFuncInter obj = new MyFuncInter(){
    public T0 m1(T1 a, T2 b){...}
};
```

LAMBDA

```
MyFuncInter obj = (a, b) -> {...};
```

(a1, a2...) \rightarrow **{code block}**

Parameters

0

()

1

(a1) or a1

N

(a1, a2...)

Body

N

{multi code lines}

1

{one code line}

or

one code line

```
list.forEach((item) -> {  
    System.out.println(item.getName());  
});
```



```
list.forEach(item -> System.out.println(item.getName()));
```

```
list.sort((o1, o2) -> {  
    return o1.getName().compareTo(o2.getName());  
});
```



```
list.sort((o1, o2) -> o1.getName().compareTo(o2.getName()));
```



DEMO



STREAM API

- ❑ Stream (giới thiệu từ Java 8) bao bọc bên trong nó một tập hợp (collection) hoặc mảng (array).
- ❑ Stream cung cấp các phương thức tăng cường xử lý các phần tử bên trong nó với kỹ thuật lập trình Lambda.
- ❑ Các operations thường dùng
 - ❖ Duyệt: ***forEach()***
 - ❖ Lọc: ***filter()***
 - ❖ Chuyển đổi: ***map()***, ***mapToDouble()***, ***mapToInt()***, ***mapToLong()***
 - ❖ Tích lũy: ***reduce()***
 - ❖ Tổng hợp: ***sum()***, ***count()***, ***min()***, ***max()***, ***average()***
 - ❖ Kiểm tra: ***allMatch()***, ***anyMatch()***, ***noneMatch()***...
 - ❖ ...

```
Staff nv1 = new Staff("Tuấn", 100);  
Staff nv2 = new Staff("Hương", 200);  
Staff nv3 = new Staff("Hạnh", 150);  
List<Staff> list = Arrays.asList(nv1, nv2, nv1);
```

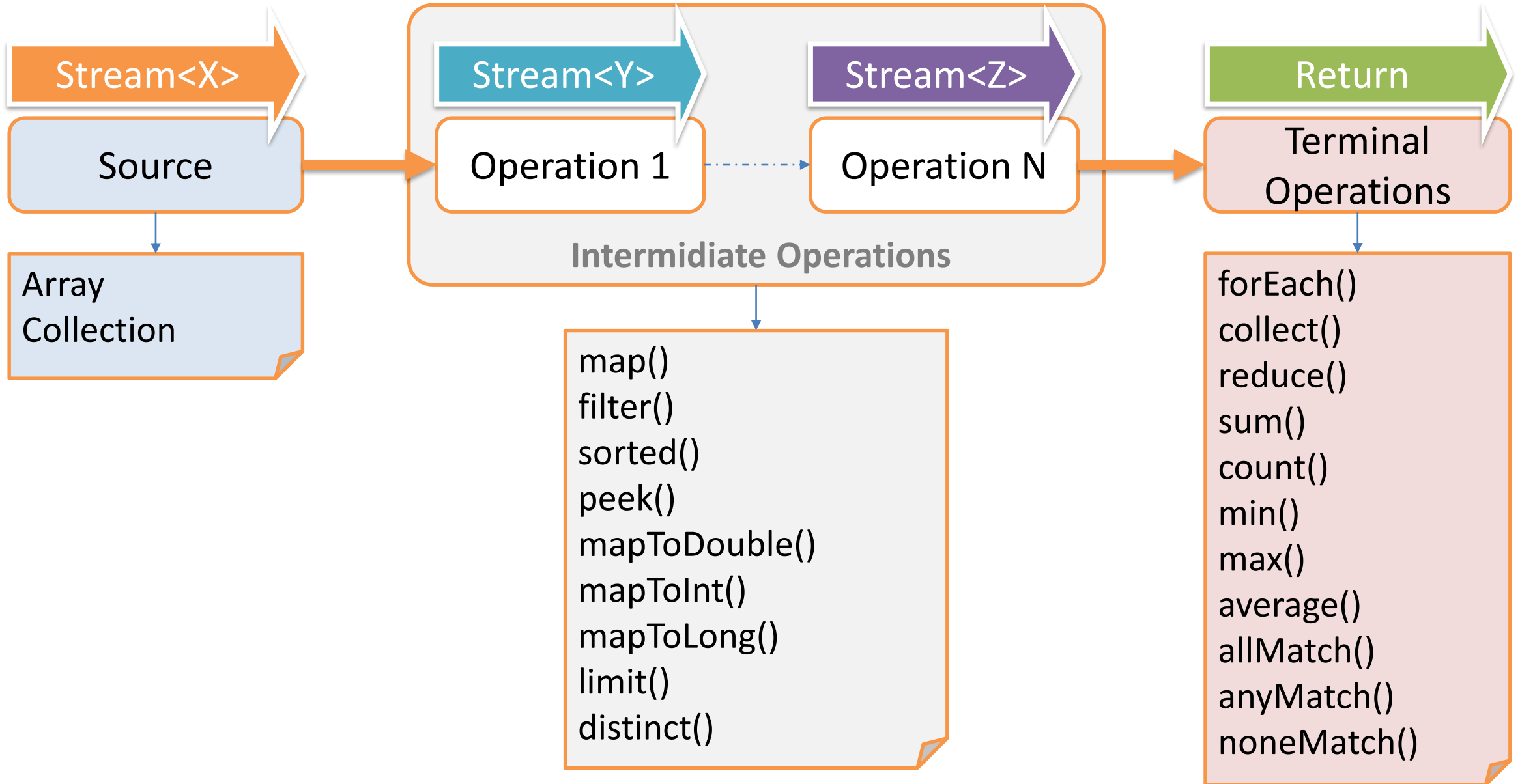
// TẠO MỚI

```
Stream<Staff> s1 = Stream.of(nv1, nv2, nv3);
```

// TẠO TỪ COLLECTION

```
Stream<Staff> s2 = list.stream();
```

STREAM PROCESSING OPERATIONS



```
List<Integer> list = Arrays.asList(2, 1, 3, 7, 8, 4, 5);
```

```
double a = list.stream()
```

Stream<Integer>

INTERMEDIATE
OPERATIONS

```
→ filter(i -> i % 2 == 0)
```

Stream<Integer>

```
→ peek(System.out::println)
```

Stream<Integer>

```
→ map(n -> Math.sqrt(n))
```

Stream<Double>

```
→ mapToDouble(d -> d)
```

DoubleStream

```
→ average().getAsDouble();
```

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

STREAM<T>.FOREACH(T -> {...})

- ❑ Stream<T> *forEach*(**item** -> {...}) được sử dụng để duyệt các phần tử trong Stream

```
// TĂNG LƯƠNG 15%
```

```
list.stream().forEach(staff -> staff.setSalary(staff.getSalary() * 1.15));
```

```
// TĂNG LƯƠNG 15% VÀ CHUYỂN TÊN SANG IN HOA
```

```
list.stream().forEach(staff -> {  
    staff.setSalary(staff.getSalary() * 1.15);  
    staff.setName(staff.getName().toUpperCase());  
});
```

STREAM<T>.FILTER(T -> BOOLEAN)

- ❑ Stream<T>.**filter**(a -> boolean) được sử dụng để lọc lấy các phần tử thỏa mãn điều kiện nào đó.

```
// HIỂN THỊ TÊN CÁC NHÂN VIÊN CÓ LƯƠNG DƯỚI 50 USD  
list.stream().filter(staff -> staff.getSalary() < 50)  
    .forEach(staff -> System.out.println(staff.getName()));
```

```
// LỌC LẤY CÁC NHÂN VIÊN HỌ NGUYỄN  
List<Staff> staffs = list.stream()  
    .filter(staff -> staff.getName().startsWith("Nguyễn "))  
    .collect(Collectors.toList());
```

- ❑ Stream<T>.**collect**(Collector) thực hiện việc thu thập các phần tử kết quả trong Stream.

- ❑ Stream<T>.**map**(T -> R) được sử dụng để chuyển đổi một phần tử đầu vào (T) thành một phần tử đầu ra (R)

```
// THU THẬP TÊN NHÂN VIÊN
```

```
List<String> names = list.stream()  
    .map(staff -> staff.getName())  
    .collect(Collectors.toList());
```

```
// THU THẬP TÊN NHÂN VIÊN CÓ LƯƠNG DƯỚI 50 USD
```

```
List<Double> sals = list.stream()  
    .filter(staff -> staff.getSalary() < 50)  
    .map(staff -> staff.getSalary())  
    .collect(Collectors.toList());
```

STREAM<T>.REDUCE(T, (T, T) -> T)

- ❑ Stream<T>.**reduce**(initial, **(temp, item) -> newTemp**) được sử dụng để tính giá trị tích lũy từ các phần tử trong Stream.

// TÍNH TỔNG THUẾ THU NHẬP

```
double incomeTax = list.stream()
    .map(staff -> staff.getSalary())
    .reduce(0.0, (subtotal, salary) -> subtotal + salary * 0.1);
```

// TÌM NHÂN VIÊN CÓ LƯƠNG THẤP NHẤT

```
Staff result = list.stream().reduce(list.get(0), (min, staff) -> {
    if(staff.getSalary() < min.getSalary()) {
        return staff;
    }
    return min;
});
```


STREAM<T>.<X>MATCH(T -> BOOLEAN)

❑ **allMatch()**, **anyMatch()** và **noneMatch()** được sử dụng để kiểm tra các phần tử bên trong Stream có thỏa mãn điều kiện nào đó hay không?

// TẤT CẢ NHÂN VIÊN ĐỀU CÓ LƯƠNG DƯỚI 50 USD

```
if(list.stream().allMatch(staff -> staff.getSalary() < 50 ) {...})
```

// ÍT NHẤT MỘT NHÂN VIÊN CÓ LƯƠNG DƯỚI 50 USD

```
if(list.stream().anyMatch(staff -> staff.getSalary() < 50 ) {...})
```

// KHÔNG MỘT NHÂN VIÊN NÀO CÓ LƯƠNG DƯỚI 50 USD

```
if(list.stream().noneMatch(staff -> staff.getSalary() < 50 ) {...})
```

❑ Ngoại trừ **count()** thì **sum()**, **min()**, **max()**, **average()** đều được thực hiện trên các số. Vì vậy cần chuyển đổi sang **<Number>Stream** trước khi thực hiện.

❖ **Stream<T>.mapToDouble(): DoubleStream**

❖ **Stream<T>.mapToInt(): IntegerStream**

❖ **Stream<T>.mapToLong(): LongStream**

// SỐ PHẦN TỬ

```
long count = list.stream().count();
```

// TỔNG GIÁ TRỊ CÁC PHẦN TỬ

```
double total = list.stream()  
    .mapToDouble(staff -> staff.getSalary()).sum();
```

// PHẦN TỬ CÓ GIÁ TRỊ NHỎ NHẤT

```
double min = list.stream()  
    .mapToDouble(staff -> staff.getSalary()).min().orElse(0);
```

- ❑ Biểu thức Lambda có thể tham chiếu đến phương thức cùng cú pháp hoặc gọi phương thức cùng kiểu.

```
Stream.of("tèo", "phèo", "kèo")  
    .forEach(s -> System.out.println(s)); ↔ Stream.of("tèo", "phèo", "kèo")  
    .forEach(System.out::println);
```

```
Stream.of(5, 7, 9)  
    .map(n -> Math.sqrt(n)) ↔ Stream.of(5, 7, 9)  
    .forEach(m -> System.out.println(m)); .map(Math::sqrt)  
    .forEach(System.out::println);
```

```
Stream.of("tèo", "phèo", "kèo")  
    .map(s -> s.toUpperCase()) ↔ Stream.of("tèo", "phèo", "kèo")  
    .forEach(ss -> System.out.println(ss)); .map(String::toUpperCase)  
    .forEach(System.out::println);
```

{
[

}
]

DEMO





JSON

- ❑ *JavaScript Object Notation (JSON) is a **standard text-based format for representing structured data based on JavaScript object syntax**. It is commonly used for transmitting data in web applications.*
- ❑ JSON là chuẩn mô tả dữ liệu theo cú pháp đối tượng JavaScript nhằm lưu trữ và trao đổi giữa các ứng dụng.
- ❑ Cú pháp (key=value):
 - ❖ Key là chuỗi (“”)
 - ❖ Value có thể là
 - Chuỗi (string)
 - Số (number)
 - Boolean
 - Mảng (array)
 - JSON (object)

```
{  
  "name": "Nguyễn Văn Tèo",  
  "marks": 7.5,  
  "gender": true,  
  "contacts": {  
    "email": "teonv@fpt.edu.vn",  
    "phone": "0913745789"  
  },  
  "subjects": ["COM107", "WEB201"]  
}
```


WORKING WITH JSON IN JAVASCRIPT?

```
var sv = {  
  "name": "Nguyễn Văn Tèo",  
  "marks": 7.5,  
  "gender": true,  
  "contacts": {  
    "email": "teonv@fpt.edu.vn",  
    "phone": "0913745789"  
  },  
  "subjects": ["COM107", "WEB201"]  
}
```

// TRUY XUẤT

```
console.log(sv['name'], sv.name);  
sv.contacts.email = 'chipheo@gmail.com';  
sv.subjects.push('WEB205');
```

// DUYỆT

```
for(var key in sv){  
  console.log(key, sv[key]);  
}
```

// CHUYỂN ĐỔI

```
var s = JSON.stringify(sv);  
var sv = JSON.parse(s);
```

{
[

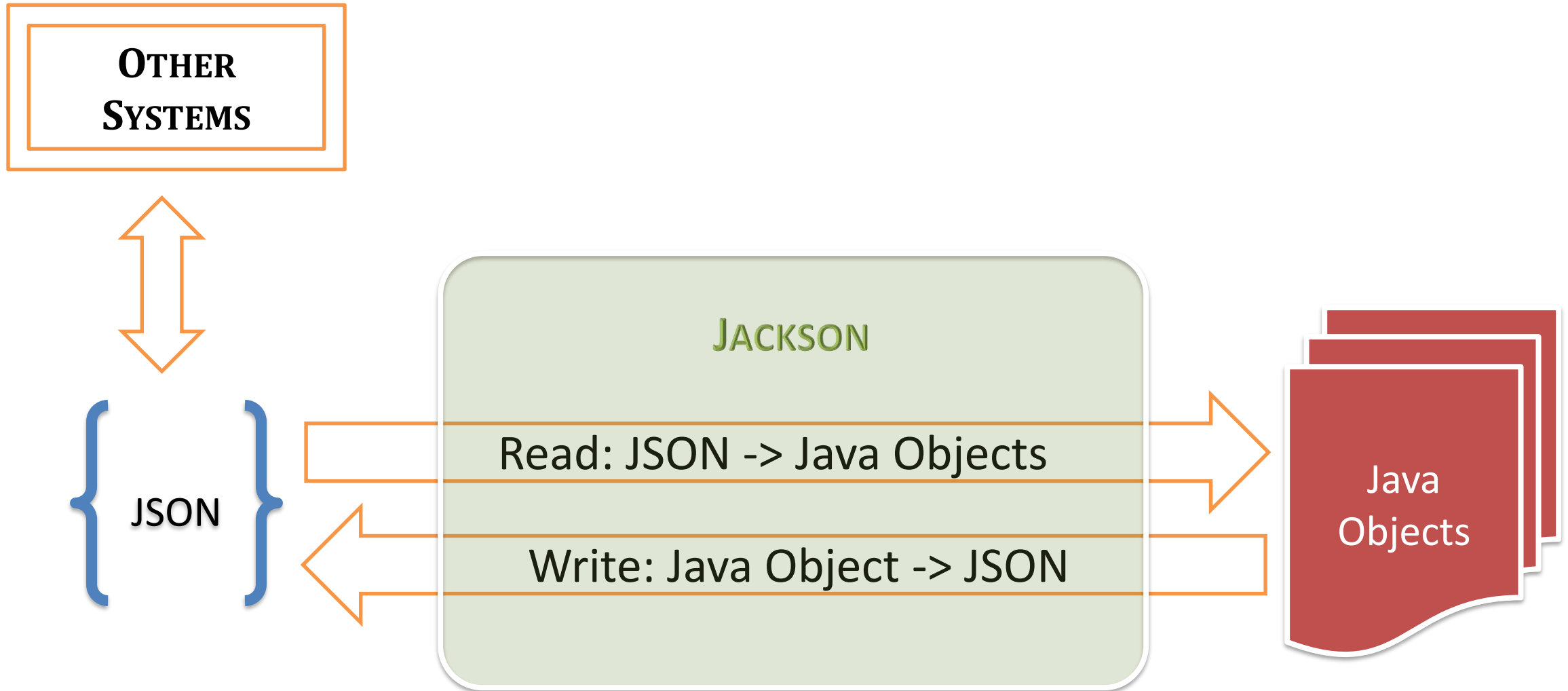
}
]

DEMO





JACKSON



❑ Reading: **JSON String -> Java Object**

❖ *readTree*(source): JsonNode

❖ *readValue*(source, Class<T>): T

➤ Source:

✓ String/byte[]

✓ Reader/InputStream/File/URL

❑ Writing: **Java Object -> JSON String**

❖ *writeValueAsBytes*(Object): byte[]

❖ *writeValueAsString*(Object): String

❖ *writeValue*(source, Object)

➤ Source

✓ Writer/OutputStream/File

❖ *writerWithDefaultPrettyPrinter().writeValueAsString*(Object)

```
String json = "{\r\n" +  
    "  \"name\": \"Nguyễn Văn Tèo\", \r\n" +  
    "  \"marks\": 7.5, \r\n" +  
    "  \"gender\": true, \r\n" +  
    "  \"contacts\": {\r\n" +  
    "    \"email\": \"teonv@fpt.edu.vn\", \r\n" +  
    "    \"phone\": \"0913745789\" \r\n" +  
    "  }, \r\n" +  
    "  \"subjects\": [\"COM107\", \"WEB201\", \"SOF307\"] \r\n" +  
    "}";
```

ObjectMapper mapper = **new** ObjectMapper();

JsonNode node = mapper.*readTree*(json);



JSON => JAVA OBJECT

READING: JSON STRING -> JSONNode

```
File source = new File("data.json");  
// File source = new URL("https://poly.web.app/data.json");  
  
ObjectMapper mapper = new ObjectMapper();  
JsonNode node = mapper.readTree(source);
```

data.json

```
{  
  "name": "Nguyễn Văn Tèo",  
  "marks": 7.5,  
  "gender": true,  
  "contacts": {  
    "email": "teonv@fpt.edu.vn",  
    "phone": "0913745789"  
  },  
  "subjects": ["COM107", "WEB201"]  
}
```

```
String name = node.get("name").asText();
double marks = node.get("marks").asDouble();
boolean gender = node.get("gender").asBoolean();
String email = node.get("contacts").get("email").asText();
String phone = node.get("contacts").get("phone").asText();
// ĐỌC DỮ LIỆU TỪ MẢNG
node.get("subjects").iterator().forEachRemaining(sub -> {
    String subject = sub.asText();
});
```

- ❑ Đọc một node: *JsonNode.get(key): JsonNode*
- ❑ Chuyển đổi kiểu dữ liệu thích hợp: *JsonNode.asType(): Type*
- ❑ Đọc mảng: *JsonNode.iterator(): Iterator<JsonNode>*

{
[

}
]

DEMO



READING: JSON STRING -> MAP<STRING, OBJECT>

```
ObjectMapper mapper = new ObjectMapper();  
Map<String, ?> map = mapper.readValue(json, Map.class);
```

```
String name = (String) map.get("name");  
boolean gender = (Boolean) map.get("gender");  
double marks = (Double) map.get("marks");
```

- ☐ String/byte[]
- ☐ File/InputStream/Reader/URL

// ĐỌC JSON LỒNG BÊN TRONG

```
Map<String, String> contacts = (Map<String, String>) map.get("contacts");  
String email = contacts.get("email");
```

// ĐỌC MẢNG

```
List<String> subjects = (List<String>) map.get("subjects");  
String com107 = subjects.get(0);
```

READING: JSON STRING -> JAVA PLAIN OBJECT

```
@Getter @Setter
public class Staff{
    String name;
    double marks;
    boolean gender;
    Contact contacts;
    List<String> subjects;
}
```

```
{
  "name": "Nguyễn Văn Tèo",
  "marks": 7.5,
  "gender": true,
  "contacts": {
    "email": "teonv@fpt.edu.vn",
    "phone": "0913745789"
  },
  "subjects": ["COM107", "WEB201"]
}
```

```
@Getter @Setter
public class Contact{
    String email;
    String phone;
    String address;
}
```

```
ObjectMapper mapper = new ObjectMapper();
Staff staff = mapper.readValue(json, Staff.class);
```



JAVA OBJECT ==> JSON

WRITING: MAP -> JSON STRING

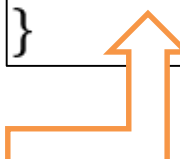
```
Map<String, String> contacts = new HashMap<>();  
contacts.put("email", "teonv@fpt.edu.vn");  
contacts.put("phone", "0913745789");
```

```
List<String> subjects = Arrays.asList("COM107", "WEB201");
```

```
Map<String, Object> staff = new HashMap<>();  
staff.put("name", "Nguyễn Văn Tèo");  
staff.put("gender", true);  
staff.put("marks", 7.5);  
staff.put("contacts", contacts);  
staff.put("subjects", subjects);
```

```
ObjectMapper mapper = new ObjectMapper();  
String json = mapper.writeValueAsString(staff);
```

```
{  
  "name": "Nguyễn Văn Tèo",  
  "marks": 7.5,  
  "gender": true,  
  "contacts": {  
    "email": "teonv@fpt.edu.vn",  
    "phone": "0913745789"  
  },  
  "subjects": ["COM107", "WEB201"]  
}
```



WRITING: STAFF -> JSON STRING

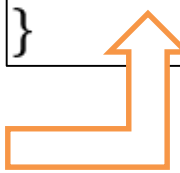
```
Contact contacts = new Contact();  
contacts.setEmail("teonv@fpt.edu.vn");  
contacts.setPhone("0913745789");
```

```
List<String> subjects = Arrays.asList("COM107", "WEB201");
```

```
Staff staff = new Staff();  
staff.setName("Nguyễn Văn Tèo");  
staff.setGender(true);  
staff.setMarks(7.5);  
staff.setContacts(contacts);  
staff.setSubjects(subjects);
```

```
ObjectMapper mapper = new ObjectMapper();  
String json = mapper.writeValueAsString(staff);
```

```
{  
  "name": "Nguyễn Văn Tèo",  
  "marks": 7.5,  
  "gender": true,  
  "contacts": {  
    "email": "teonv@fpt.edu.vn",  
    "phone": "0913745789"  
  },  
  "subjects": ["COM107", "WEB201"]  
}
```



{
[

}
]

DEMO



- ✓ Lambda Expression
- ✓ Stream API
 - ✓ Filter()
 - ✓ Map()
 - ✓ Reduce()
 - ✓ allMatch()/anyMatch()/noneMatch()
- ✓ JSON
- ✓ Jackson API
 - ✓ JsonNode
 - ✓ Jackson with Map
 - ✓ Jackson with Plain Object





FPT Education
FPT POLYTECHNIC
Thank you