VIETNAM AUSTRALIA Vocational College

# Slide-2.3: Optional, Predicate, String Joiner, Sorting

*Mentor: Nguyễn Bá Minh Đạo*

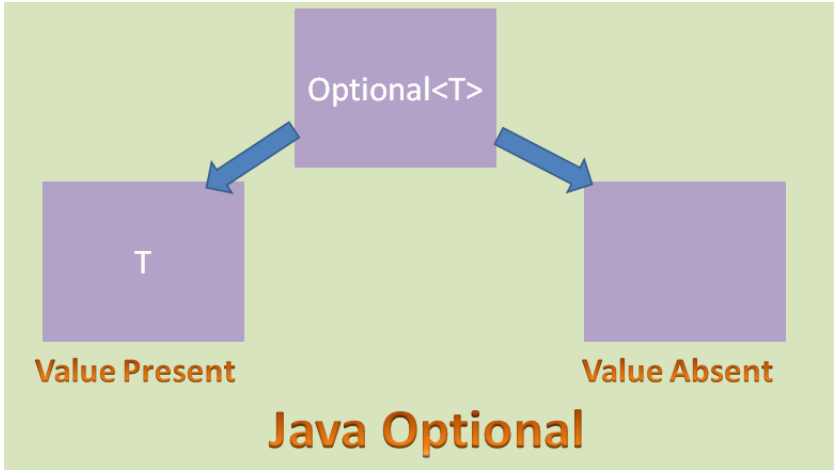
# ***Nội dung***

1. Optional trong Java 8
2. Predicate trong Java 8
3. String Joiner trong Java 8
4. Sorting trong Java 8

# Optional trong Java 8

## Optional<T> trong Java 8

- Trong Java 8, chúng ta có một lớp **Optional<T>** mới được giới thiệu trong gói **java.util**
- Được sử dụng để kiểm tra xem một biến có giá trị tồn tại giá trị hay không.
- Ưu điểm chính của cấu trúc này là không có quá nhiều kiểm tra null và tránh lỗi NullPointerException (**NPE**) lúc runtime.
- Giống như **Collection** và **Array**, **Optional** cũng là một vùng chứa chứa nhiều giá trị nhất một giá trị. Vùng này có thể chứa bất kỳ đối tượng **T** nào.



# Optional trong Java 8

## Một số phương thức của Optional<T>

| Phương thức | Mô tả |
|---|---|
| **static <T> Optional<T> empty()** | Trả về một **Optional** instance rỗng |
| **static <T> Optional<T> of(T value)** | Trả về đối tượng **Optional** kiểu T chứa giá trị của value |
| **static <T> Optional<T> ofNullable(T value)** | Trả về một **Optional** chứa giá trị được truyền vào nếu khác null, ngược lại sẽ trả về một Optional rỗng |
| **T get()** | Nếu như có giá trị trong **Optional**, sẽ trả về giá trị đó, ngược lại ném ra **NoSuchElementException** nếu như đối tượng rỗng |
| **void ifPresent(Consumer<? Super T> consumer)** | Nếu như **Optional** đang chứa giá trị, nó sẽ áp dụng **consumer** được truyền vào cho giá trị của nó. Ngược lại, không làm gì cả. |

# Optional trong Java 8

## Một số phương thức của Optional<T>

| Phương thức | Mô tả |
|---|---|
| **boolean ifPresent()** | Phương thức này được sử dụng để kiểm tra xem trong đối tượng **Optional** có đang chứa giá trị hay không. Giá trị trả về là **true** nếu có giá trị và ngược lại trả về **false** |
| **orElse(T other)** | Trả về giá trị của đối tượng **Optional** nếu có, ngược lại, sẽ trả về đối tượng other mà bạn đã truyền vào phương thức này. |
| **T orElseGet(Supplier<? extends T> other)** | Trả về giá trị nếu tồn tại, ngược lại nó sẽ gọi other mà bạn đã truyền vào sau đó trả về kết quả của **Supplier**. |
| **<U>Optional<U> map(Function<? super T,? extends U> mapper)** | Nếu như có giá trị nó sẽ áp dụng Function lên giá trị đó và nếu kết quả sau khi áp dụng Function khác null thì nó sẽ trả về một Optional chứa giá trị của kết quả sau khi áp dụng Function |

# Optional trong Java 8

## Một số phương thức của Optional<T>

| Phương thức | Mô tả |
|---|---|
| **<U> Optional<U> flatMap(Function<? super T,Optional<U>> mapper)** | Nếu như có giá trị hiện diện trong Optional, nó sẽ áp dụng Function được truyền vào cho nó và trả về một Optional với kiểu U, ngược lại thì sẽ return về một empty Optional. |
| **Optional<T> filter(Predicate<? super <T> predicate)** | Nếu như có giá trị hiện diện trong đối tượng Optional này và giá trị của nó khớp với Predicate truyền vào, nó sẽ trả về một Optional chứa giá trị đó, mặc khác sẽ trả về một Optional rỗng. |
| **<X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier)** | Nếu như đối tượng Optional có giá trị tồn tại thì nó sẽ trả về giá trị đó, ngược lại sẽ ném ra một Exception do chúng ta định nghĩa bởi Supplier đã truyền vào. |

# Optional trong Java 8

## Khởi tạo đối tượng Optional<T>

- Tạo một đối tượng Optional rỗng (empty)

```
Optional<String> optional = Optional.empty();
```

- Tạo một đối tượng Optional chứa giá trị non-null

```
String str = “javacoder”;
Optional<String> optional = Optional.of(str);
```

- Tạo một đối tượng Optional chứa giá trị cụ thể hoặc giá trị rỗng (empty) nếu tham số là null

```
Optional<String> optional = Optional.ofNullable(obj);
```

# Optional trong Java 8

## *Ví dụ sử dụng get(), isPresent(), ifPresent()*

- Trước phiên bản Java 8 chúng ta cần phải kiểm tra đối tượng **!= null** trước khi sử dụng.

- Với Java 8, chúng ta có thể sử dụng phương thức **isPresent()** của **Optional** để kiểm tra tồn tại giá trị trước khi sử dụng.

```
class Student {
    String name;
}

public class OptionalExample01 {

    public void preJava8() {
        Student student = getStudent();
        if (student != null) {
            System.out.println(student.name);
        }
    }

    public void java8() {
        Student student = getStudent();
        Optional<Student> opt = Optional.of(student);
        if (opt.isPresent()) {
            System.out.println(opt.get().name);
        }
        // opt.ifPresent(s -> System.out.println(s.name));
    }

    private Student getStudent() {
        Student student = new Student();
        student.name = "javacoder";
        return null;
    }
}
```

# Optional trong Java 8

## Ví dụ sử dụng orElse(), orElseGet()

- **orElse()** và **orElseGet()** trả về giá trị của đối tượng **Optional** nếu nó không rỗng, nếu đối tượng rỗng thì trả về giá trị mặc định được truyền cho phương thức này.
- Sự khác nhau giữa **orElse()** và **orElseGet()**:
  - **orElse(T other)**: phương thức **orElse()** luôn được thực thi cho dù đối tượng **Optional** có rỗng hay không. Và chỉ sử dụng phương thức này khi giá trị của nó là một constant, nếu cần xử lý **heavy logic** để có giá trị **default** thì chúng ta nên sử dụng **orElseGet()** để có performance tốt hơn.
  - **orElseGet(Supplier<? Extends T> other)**: tương tự như **orElse()**. Tuy nhiên, nó chỉ được gọi khi đối tượng **Optional** là rỗng.

# Optional trong Java 8

## Ví dụ sử dụng orElse(), orElseGet()

- **Ví dụ** nếu mong muốn chương trình trả về một List rỗng nếu không tồn tại giá trị. Hãy xem cách viết sử dụng phiên bản trước Java 8 và **Optional** của Java 8 như sau:

```
public class OptionalExample02 {

    public static void main(String[] args) {
        // Pre java 8
        List<String> list = getList();
        List<String> listOpt = list != null ? list : new ArrayList<>();

        // Java 8
        List<String> listOpt2 = getList2().orElse(new ArrayList<>());
        List<String> listOpt3 = getList2().orElseGet(() -> new ArrayList<>());
    }

    private static List<String> getList() {
        return null;
    }

    private static Optional<List<String>> getList2() {
        return Optional.ofNullable(null);
    }
}
```

# Optional trong Java 8

## Ví dụ sử dụng map()

- Nếu có một giá trị, áp dụng hàm ánh xạ được cung cấp cho nó. Nếu kết quả **!= null**, trả về một **Optional** với kết quả. Ngược lại thì trả về một **Optional** rỗng.

```
class User {

    private Address address;

    public Address getAddress() {
        return address;
    }
}

class Address {

    private String street;

    public String getStreet() {
        return street;
    }
}
```

```
public class OptionalExample03 {
    public String getStreetPreJava8() {
        User user = getUser();
        if (user != null) {
            Address address = user.getAddress();
            if (address != null) {
                String street = address.getStreet();
                if (street != null) {
                    return street;
                }
            }
        }
        return "not specified";
    }

    public String getStreetJava8() {
        Optional<User> user = Optional.ofNullable(getUser());
        return user.map(User::getAddress)
                .map(Address::getStreet)
                .orElse("not specified");
    }

    private static User getUser() {
        return null;
    }
}
```

# Optional trong Java 8

## Ví dụ sử dụng flatMap()

- Sử dụng **map()** có thể tồn tại các Optional lồng nhau. Sử dụng **flatMap()**, các **Optional** lồng nhau sẽ được gộp (flat) lại một **Optional** duy nhất.

```
public class OptionalExample04 {
    public static void main(String[] args) {
        Optional<String> optionalOf = Optional.of("welcome to javacoder");
        Optional<String> optionalEmpty = Optional.empty();

        System.out.println(optionalOf.map(String::toLowerCase));
        System.out.println(optionalEmpty.map(String::toLowerCase));

        Optional<Optional<String>> multiOptional = Optional.of(Optional.of("javacoder"));

        System.out.println("Value of Optional object: " + multiOptional);
        System.out.println("Optional.map: " + multiOptional.map(gender -> gender.map(String::toUpperCase)));
        System.out.println("Optional.flatMap: " + multiOptional.flatMap(gender -> gender.map(String::toUpperCase)));
    }
}
```

- Như bạn thấy **Optional<Optional<String>>** khi sử dụng **flatMap()** kết quả sẽ là **Optional<String>**, còn **map()** vẫn là **Optional<Optional<String>>**.

# Optional trong Java 8

## Ví dụ sử dụng filter(), orElseThrow()

- Nếu có giá trị trong đối tượng **Optional** này và giá trị của nó khớp với **Predicate** truyền vào, nó sẽ trả về một **Optional** chứa giá trị đó, mặc khác sẽ trả về một **Optional** rỗng.

```
public class OptionalExample05 {
    public static void main(String[] args) {
        Optional<String> me = Optional.of("javacoder");
        Optional<String> emptyOptional = Optional.empty();

        //Filter on Optional
        System.out.println(me.filter(g -> g.equals("JAVACODER"))); //Optional.empty
        System.out.println(me.filter(g -> g.equalsIgnoreCase("javacoder"))); //Optional[javacoder]
        System.out.println(emptyOptional.filter(g -> g.equalsIgnoreCase("javacoder"))); //Optional.empty
    }
}
```

- Nếu như đối tượng Optional có giá trị tồn tại thì nó sẽ trả về giá trị đó, ngược lại sẽ ném ra một **Exception** do chúng ta định nghĩa bởi **Supplier** đã truyền vào.

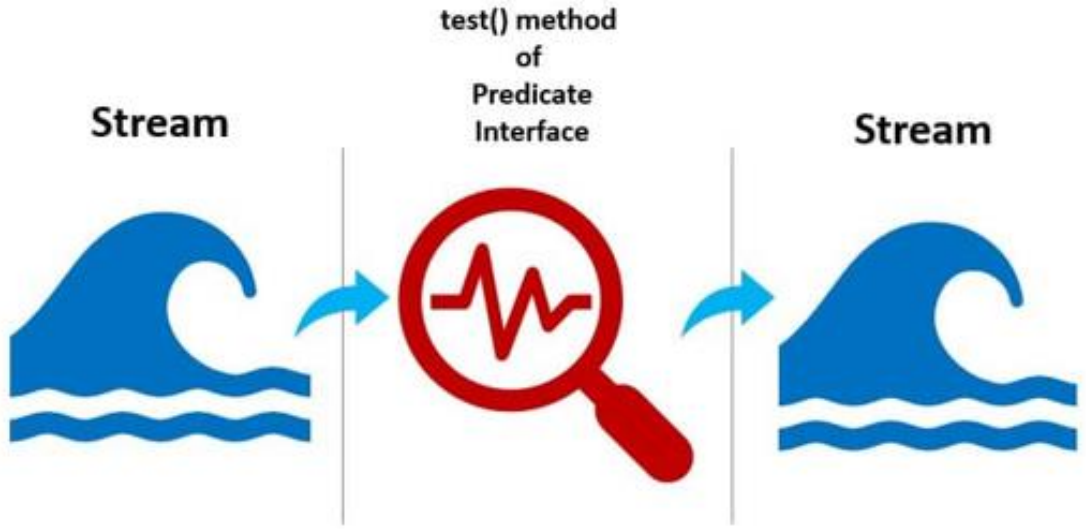
```
public class OptionalExample06 {

    public static void main(String[] args) {
        Optional<String> shouldNotBeEmpty = Optional.empty();
        shouldNotBeEmpty.orElseThrow(() -> new IllegalStateException("This should not happen!!!"));
    }
}
```

```
Exception in thread "main" java.lang.IllegalStateException: This should not happen!!!
        at com.vu.optional.OptionalExample06.lambda$0(OptionalExample06.java:9)
        at java.base/java.util.Optional.orElseThrow(Optional.java:403)
        at com.vu.optional.OptionalExample06.main(OptionalExample06.java:9)
```

# Predicate trong Java 8

## Giới thiệu Predicate<T>

- **Predicate<T>** là một **Functional Interface** và do đó nó có thể được sử dụng với **Lambda Expression** hoặc **Method Reference** cho một mục đích cụ thể nào đó.
- **Predicate<T>** sẽ trả về giá trị true/false của một tham số kiểu **T** mà bạn đưa vào có thỏa với điều kiện của **Predicate** đó hay không, cụ thể là điều kiện được viết trong phương thức **test()**.



# Predicate trong Java 8

## Giới thiệu Predicate<T>

❑ Interface **Predicate** được khai báo trong package **java.util.function** như sau:

```
@FunctionalInterface
public interface Predicate<T> {

    /**
     * Evaluates this predicate on the given argument.
     *
     * @param t the input argument
     * @return {@code true} if the input argument matches the predicate,
     * otherwise {@code false}
     */
    boolean test(T t);
```

❑ Trong đó:

➢ **boolean test(T t)**: là một phương thức trừu tượng có thể được sử dụng với **lambda expression** hoặc **method reference** cho một mục đích cụ thể nào đó.

➢ Phương thức **test()** trả về **true** nếu đối số đầu vào khớp với biến **predicate** (điều kiện kiểm tra), nếu không trả về **false**.

# Predicate trong Java 8

## Một số phương thức mặc định (default method) của Predicate<T>

- **and()**: Nó thực hiện logic AND của predicate mà nó được gọi với một biến predicate khác. Ví dụ: predicate1.and(predicate2)
- **or()**: Nó thực hiện logic OR của predicate mà nó được gọi với một biến predicate khác. Ví dụ: predicate1.or(predicate2)
- **negate()**: Nó thực hiện phủ định kết quả của biến predicate được gọi. Ví dụ: predicate1.negate()
- Ngoài ra interface Predicate còn có một phương thức tĩnh (static method) là **isEquals()**. Phương thức này được sử dụng để kiểm tra hai đối số có số bằng nhau theo Objects.equals(Object, Object)

# Predicate trong Java 8

## Ví dụ sử dụng phương thức test() của Predicate<T>

```
public class PredicateExample01 {

    public static void main(String[] args) {

        // Predicate String
        Predicate<String> predicateString = s -> {
            return s.equals("javacoder");
        };
        System.out.println(predicateString.test("javacoder")); // true
        System.out.println(predicateString.test("Java Coder")); // false

        // Predicate integer
        Predicate<Integer> predicateInt = i -> {
            return i > 0;
        };
        System.out.println(predicateInt.test(1)); // true
        System.out.println(predicateInt.test(-1)); // false
    }
}
```

# Predicate trong Java 8

## Ví dụ phương thức and(), or(), negate(), isEquals() của Predicate<T>

```
public class PredicateExample02 {

    public static void main(String[] args) {

        Predicate<String> predicate = s -> {
            return s.equals("javacoder");
        };

        // AND logical operation
        Predicate<String> predicateAnd = predicate.and(s -> s.length() == 9);
        System.out.println(predicateAnd.test("javacoder")); // true

        // OR logical operation
        Predicate<String> predicateOr = predicate.or(s -> s.length() == 9);
        System.out.println(predicateOr.test("javacoder")); // true

        // NEGATE logical operation
        Predicate<String> predicateNegate = predicate.negate();
        System.out.println(predicateNegate.test("javacoder")); // false
    }
}
```

# Predicate trong Java 8

## Ví dụ kết hợp nhiều phương thức của Predicate<T>

```
public class PredicateExample03 {

    public static void main(String[] args) {

        // Creating predicate
        Predicate<Integer> greaterThanTen = (i) -> i > 10;
        Predicate<Integer> lessThanTwenty = (i) -> i < 20;

        // Calling Predicate Chaining
        boolean result = greaterThanTen.and(lessThanTwenty).test(15);
        System.out.println(result); // true

        // Calling Predicate method
        boolean result2 = greaterThanTen.and(lessThanTwenty).negate().test(15);
        System.out.println(result2); // false
    }
}
```

# Predicate trong Java 8

## Ví dụ sử dụng Predicate<T> với Collection

- Tạo lớp **Employee** bao gồm các thông tin như bên dưới và các **Predicate** để thực thi:

```java
public class Employee {

    private Integer id;
    private Integer age;
    private String gender;
    private String firstName;
    private String lastName;

    public Employee(Integer id, Integer age, String gender, String fName, String lName) {

    @Override
    public String toString() {
        return this.id.toString() + " - " + this.age.toString();
    }

    public Integer getId() {

    public void setId(Integer id) {

    public Integer getAge() {

    public void setAge(Integer age) {

    public String getGender() {

    public void setGender(String gender) {

    public String getFirstName() {

    public void setFirstName(String firstName) {

    public String getLastName() {

    public void setLastName(String lastName) {

}
```

```java
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class EmployeePredicates {

    // Tất cả Employee có tuổi > 21 và có giới tính là Male
    public static Predicate<Employee> isAdultMale() {
        return p -> p.getAge() > 21 && p.getGender().equalsIgnoreCase("M");
    }

    // Tất cả Employee có tuổi > 18 và có giới tính là Female
    public static Predicate<Employee> isAdultFemale() {
        return p -> p.getAge() > 18 && p.getGender().equalsIgnoreCase("F");
    }

    // Tất cả Employee lớn hơn số tuổi được truyền vào
    public static Predicate<Employee> isAgeMoreThan(Integer age) {
        return p -> p.getAge() > age;
    }

    // Lấy danh sách Employee thỏa mãn predicate được truyền vào
    public static List<Employee> filterEmployees( //
            List<Employee> employees, Predicate<Employee> predicate) {
        return employees.stream().filter(predicate).collect(Collectors.<Employee>toList());
    }
}
```

# Predicate trong Java 8

## Ví dụ sử dụng Predicate<T> với Collection

❑ Chúng ta sẽ bắt đầu apply các **Predicate** trên như sau:

```
public class EmployeePredicateTest {

    public static void main(String[] args) {

        List<Employee> employees = Arrays.asList( //
                new Employee(1, 23, "M", "Rick", "Beethovan"), //
                new Employee(2, 13, "F", "Martina", "Hengis"), //
                new Employee(3, 43, "M", "Ricky", "Martin"), //
                new Employee(4, 26, "M", "Jon", "Lowman"), //
                new Employee(5, 19, "F", "Cristine", "Maria"), //
                new Employee(6, 15, "M", "David", "Feezor"), //
                new Employee(7, 68, "F", "Melissa", "Roy"), //
                new Employee(8, 79, "M", "Alex", "Gussin"), //
                new Employee(9, 15, "F", "Neetu", "Singh"), //
                new Employee(10, 45, "M", "Naveen", "Jain") //
        );

        // Lấy danh sách Employee nam 21+
        System.out.println(filterEmployees(employees, isAdultMale()));

        // Lấy danh sách Employee nữ 18+
        System.out.println(filterEmployees(employees, isAdultFemale()));

        // Lấy danh sách Employee > 35 tuổi
        System.out.println(filterEmployees(employees, isAgeMoreThan(35)));

        // Lấy danh sách Employee <= 35 tuổi
        System.out.println(filterEmployees(employees, isAgeMoreThan(35).negate()));
    }
}
```

# Predicate trong Java 8

## Ví dụ sử dụng Predicate<T> với các lớp cho kiểu dữ liệu nguyên thủy

- Java 8 cung cấp một số **Interface Predicate** cho các **wrapper class** của kiểu dữ liệu nguyên thủy như sau:
  - **IntPredicate**: chấp nhận một giá trị kiểu **Int** duy nhất.
  - **LongPredicate**: chấp nhận một giá trị kiểu **Long** duy nhất.
  - **DoublePredicate**: chấp nhận một giá trị kiểu **Double** duy nhất.

```
public class PredicateExample04 {

    public static void main(String[] args) {

        System.out.print("IntPredicate: ");
        int[] intNumbers = { 3, 5, 6, 2, 1 };
        IntPredicate intPredicate = i -> i > 5;
        Arrays.stream(intNumbers).filter(intPredicate).forEach(System.out::println);

        System.out.print("\nLongPredicate: ");
        long[] longNumbers = { 3, 5, 6, 2, 1 };
        LongPredicate longPredicate = l -> l > 5;
        Arrays.stream(longNumbers).filter(longPredicate).forEach(System.out::println);

        System.out.print("\nDoublePredicate: ");
        double[] dbNumbers = { 3.2, 5.0, 6.3, 2.5, 1.0 };
        DoublePredicate dbPredicate = d -> d > 5;
        Arrays.stream(dbNumbers).filter(dbPredicate).forEach(System.out::println);
    }
}
```

# Predicate trong Java 8

## Ví dụ sử dụng Predicate 2 đối số với BiPredicate

- Như đã học **Predicate** chỉ chấp nhận **1** đối số đầu vào và trả về 1 giá trị true/false, để có thể sử dụng **Predicate** với **2** đối số đầu vào chúng ta sử dụng Interface **BiPredicate**

```
42 @FunctionalInterface
43 public interface BiPredicate<T, U> {
44
45     /**
46      * Evaluates this predicate on the given arguments.
47      *
48      * @param t the first input argument
49      * @param u the second input argument
50      * @return {@code true} if the input arguments match the predicate,
51      * otherwise {@code false}
52      */
53     boolean test(T t, U u);
```

- Về cơ bản, interface **BiPredicate** không khác biệt so với **Predicate**, ngoại trừ việc nó chấp nhận 2 đối số đầu vào.

```
public class BiPredicateExample {

    public static void main(String[] args) {

        BiPredicate<Integer, String> condition = (i, s) -> i > 2 && s.startsWith("J");
        System.out.println(condition.test(5, "Java")); // true
        System.out.println(condition.test(2, "Javascript")); // false
        System.out.println(condition.test(1, "C#")); // false
    }
}
```

# String Joiner trong Java 8

## Sử dụng StringBuilder/StringBuffer trước Java 8

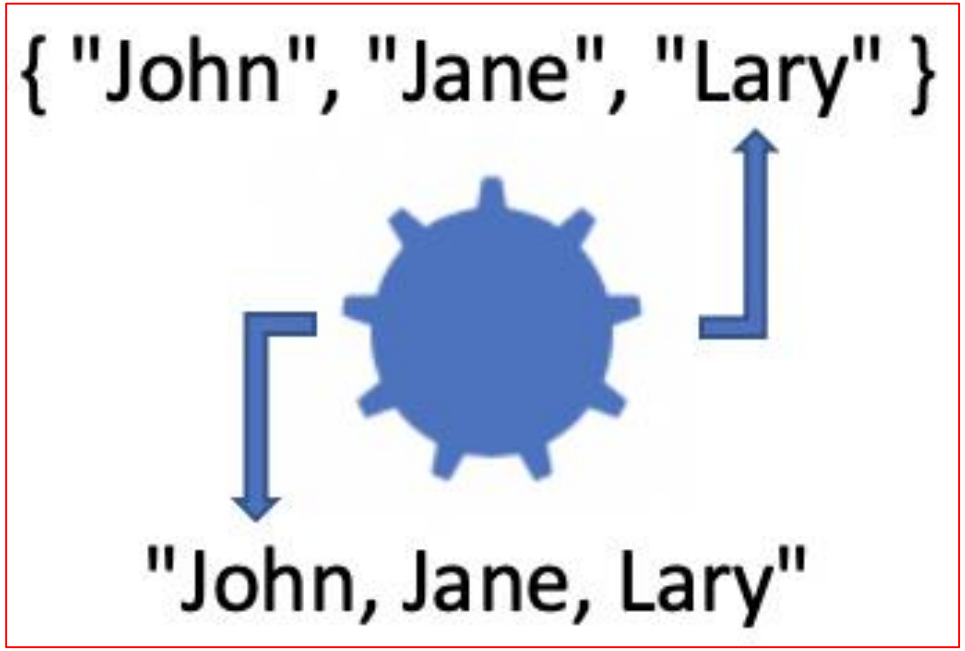
- Trước Java 8, để ghép các chuỗi với dấu phân cách, chúng ta phải lặp qua các phần tử của một mảng hoặc danh sách và sử dụng **StringBuilder/StringBuffer** để lưu giữ.

```
public class StringBuilderStringBufferExample {

    public static void main(String[] args) {
        final String DELIMITER = ", ";
        String[] arr = { "one", "two", "three" };
        int numOfElements = arr.length;
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < numOfElements; i++) {
            sb.append(arr[i]);
            if (i < numOfElements - 1) {
                sb.append(DELIMITER);
            }
        }
        System.out.println(sb.toString()); // one, two, three
    }
}
```

# String Joiner trong Java 8

## Sử dụng StringJoiner trong Java 8

- Java 8 đã thêm một lớp final **StringJoiner** trong gói **java.util**.
- **StringJoiner** được sử dụng để xây dựng một chuỗi ký tự được phân tách bằng dấu phân cách như dấu phẩy (,), dấu nối (-), ...
- Bạn cũng có thể chuyển tiền tố (prefix) và hậu tố (suffix) vào chuỗi.



# String Joiner trong Java 8

## Ví dụ sử dụng StringJoiner với dấu phân cách (delimiter)

```java
public class StringJoinerExample01 {

    public static void main(String[] args) {
        StringJoiner stringJoiner = new StringJoiner(", ");
        stringJoiner.add("one");
        stringJoiner.add("two");
        stringJoiner.add("three");
        System.out.println(stringJoiner.toString()); // one, two, three
    }
}
```

## Ví dụ sử dụng StringJoiner với tiền tố (prefix) và hậu tố (suffix)

```java
public class StringJoinerExample02 {

    public static void main(String[] args) {
        StringJoiner stringJoinerWithPrefixSufix = new StringJoiner(",", "{", "}");
        stringJoinerWithPrefixSufix.add("1");
        stringJoinerWithPrefixSufix.add("2");
        stringJoinerWithPrefixSufix.add("3");
        System.out.println(stringJoinerWithPrefixSufix.toString()); // {1,2,3}
    }
}
```

## Ví dụ sử dụng phương thức static java.lang.String.join()

```java
public class StringJoinerExample03 {

    public static void main(String[] args) {
        String strWithJoiner = String.join(", ", "one", "two", "three");
        System.out.println(strWithJoiner); // one, two, three

        List<String> list= Arrays.asList("one", "two", "three");
        strWithJoiner = String.join(", ", list); // one, two, three
    }
}
```

## Ví dụ sử dụng phương thức java.util.stream.Collectors.joining()

```java
public class StringJoinerExample04 {

    public static void main(String[] args) {
        // joining with delimiter
        List<String> list = Arrays.asList("one", "two", "three");
        String collectorJoiningWithDelimiter = list.stream()
                .map(element -> element)
                .collect(Collectors.joining(", "));
        System.out.println(collectorJoiningWithDelimiter); // one, two, three

        // joining with prefix & suffix
        String collectorJoiningWithPrefixSuffix = list.stream()
                .map(element -> element)
                .collect(Collectors.joining(", ", "{", "}"));
        System.out.println(collectorJoiningWithPrefixSuffix); // {one, two, three}

    }
}
```

* **Lưu ý**: Nếu project của bạn không sử dụng phiên bản Java 8, các bạn cũng có thể sử dụng thư viện common-lang để thay thế.

# Sorting trong Java 8

## Sắp xếp mảng (Array)

❑ Để sắp xếp các phần tử của mảng, chúng ta sử dụng lớp tiện ích **Arrays.sort()**

- ➢ **Arrays.sort(arr)**: Sắp xếp tất cả các phần tử của mảng.
- ➢ **Arrays.sort(arr, fromIndex, toIndex)**: Sắp xếp một phần của mảng.
- ➢ **Arrays.parallelSort.sort(arr)**:
  - ▪ Sắp xếp tất cả các phần tử của mảng theo cách xử lý song song.
  - ▪ Phương thức này chia nhỏ một mảng thành nhiều mảng con và thực hiện sắp xếp trên các mảng con này một cách song song trên các luồng (Thread) khác nhau
  - ▪ Sau khi sắp xếp, các mảng con merge lại để có một mảng được sắp xếp hoàn chỉnh.
- ➢ **Arrays.parallelSort.sort(arr, fromIndex, toIndex)**: Sắp xếp một phần của mảng theo cách xử lý song song.

# Sorting trong Java 8

## Sắp xếp mảng (Array)

❑ Ví dụ:

```
public class SortedArrayExample {
    public static final int NUMBERS[] = { 5, 1, 2, 4, 3, 6, 7, 9, 8 };

    public static void main(String[] args) {
        // Sorting Complete Array
        int arr1[] = Arrays.copyOf(NUMBERS, NUMBERS.length);
        Arrays.sort(arr1);
        System.out.println(Arrays.toString(arr1));
        // => [1, 2, 3, 4, 5, 6, 7, 8, 9]

        // Sorting Part of an Array
        int arr2[] = Arrays.copyOf(NUMBERS, NUMBERS.length);
        Arrays.sort(arr2, 2, 5);
        System.out.println(Arrays.toString(arr2));
        // => [5, 1, 2, 3, 4, 6, 7, 9, 8]

        // Java 8 parallelSort
        int arr3[] = Arrays.copyOf(NUMBERS, NUMBERS.length);
        Arrays.parallelSort(arr3);
        System.out.println(Arrays.toString(arr3));
        // => [1, 2, 3, 4, 5, 6, 7, 8, 9]
    }
}
```

# Sorting trong Java 8

## Sắp xếp mảng (List)

❑ Để sắp xếp các phần tử của danh sách, chúng ta sử dụng lớp tiện ích **Collections.sort()**

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class SortedListExample {
    public static final List<Integer> NUMBERS = Arrays.asList( 5, 1, 2, 4, 3, 6, 7, 9, 8 );

    public static void main(String[] args) {
        // Sorting a List
        List<Integer> list1 = new ArrayList<>(NUMBERS);
        Collections.sort(list1);
        System.out.println(list1);
        // => [1, 2, 3, 4, 5, 6, 7, 8, 9]
    }
}
```

# Sorting trong Java 8

## Sắp xếp mảng (Set)

- Lớp tiện ích **Collections** chỉ hỗ trợ sắp xếp các phần tử trong một **List**. Do đó, để có thể sắp xếp được một **Set** chúng ta cần chuyển một **Set** qua một **List**, sau đó thực hiện sắp xếp danh sách này và cuối cùng thực hiện chuyển **List** về **Set**.

```
public class SortedSetExample {
    public static final List<Integer> NUMBERS = Arrays.asList( 5, 1, 2, 4, 3, 6, 7, 9, 8 );

    public static void main(String[] args) {
        // Original data
        Set<Integer> set1 = new LinkedHashSet<>(NUMBERS);

        // Convert Set to List
        List<Integer> list1 = new ArrayList<>(set1);

        // Sorting a List
        Collections.sort(list1);

        // Convert List to Set
        set1 = new LinkedHashSet<>(list1);
        System.out.println(set1);
        // => [1, 2, 3, 4, 5, 6, 7, 8, 9]
    }
}
```

# Sorting trong Java 8

## Sắp xếp mảng (Map) theo key và value

- Một **Map** gồm 2 thành phần **Key** và **Value**. Do đó, chúng ta có thể sắp xếp theo **Key** hoặc **Value** tùy theo nhu cầu sử dụng.

```java
public class SortedMapExample01 {

    public static void main(String[] args) {
        // Original data
        Map<Integer, String> map = new HashMap<>();
        map.put(44, "Four");
        map.put(22, "Two");
        map.put(33, "Three");
        map.put(55, "Five");
        map.put(11, "One");

        // Convert list to map
        List<Map.Entry<Integer, String>> list1 = new ArrayList<>(map.entrySet());

        // Create a comparator to sort by key
        Comparator<Entry<Integer, String>> keyComparator = new Comparator<Entry<Integer, String>>() {
            @Override
            public int compare(Entry<Integer, String> o1, Entry<Integer, String> o2) {
                return o1.getKey().compareTo(o2.getKey());
            }
        };

        // Sorting a List
        Collections.sort(list1, keyComparator);

        // Convert List to Map
        Map<Integer, String> sortedMap = new LinkedHashMap<>();
        for (Map.Entry<Integer, String> entry : list1) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        System.out.println("Original map: " + map);
        System.out.println("Sorted map: " + sortedMap);
    }
}
```

```java
public class SortedMapExample02 {

    public static void main(String[] args) {
        // Original data
        Map<Integer, String> map = new HashMap<>();
        map.put(44, "Four");
        map.put(22, "Two");
        map.put(33, "Three");
        map.put(55, "Five");
        map.put(11, "One");

        // Convert list to map
        List<Map.Entry<Integer, String>> list1 = new ArrayList<>(map.entrySet());

        // Create a comparator to sort by value
        Comparator<Entry<Integer, String>> valueComparator = new Comparator<Entry<Integer, String>>() {
            @Override
            public int compare(Entry<Integer, String> o1, Entry<Integer, String> o2) {
                return o1.getValue().compareTo(o2.getValue());
            }
        };

        // Sorting a List
        Collections.sort(list1, valueComparator);

        // Convert List to Map
        Map<Integer, String> sortedMap = new LinkedHashMap<>();
        for (Map.Entry<Integer, String> entry : list1) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        System.out.println("Original map: " + map);
        System.out.println("Sorted map: " + sortedMap);
    }
}
```

# Sorting trong Java 8

## Sắp xếp mảng (Map) trong Java 8

- Với Java 8, **Map** cung cấp thêm một số phương thức hỗ trợ sắp xếp theo **Key**, **Value**:
  - comparingByKey()
  - comparingByKey(Comparator<? super K> cmp)
  - comparingByValue()
  - comparingByValue(Comparator<? super V> cmp)

```
public class SortedMapExample03 {

    public static void main(String[] args) {
        // Original data
        Map<Integer, String> map = new HashMap<>();
        map.put(44, "Four");
        map.put(22, "Two");
        map.put(33, "Three");
        map.put(55, "Five");
        map.put(11, "One");

        // sort by key
        Map<Integer, String> sortedMapByKey = new LinkedHashMap<>();
        map.entrySet().stream()
                .sorted(Map.Entry.<Integer, String>comparingByKey())
                .forEachOrdered(e -> sortedMapByKey.put(e.getKey(), e.getValue()));

        // sort by value
        Map<Integer, String> sortedMapByValue = new LinkedHashMap<>();
        map.entrySet().stream()
                .sorted(Map.Entry.<Integer, String>comparingByValue())
                .forEachOrdered(e -> sortedMapByValue.put(e.getKey(), e.getValue()));

        // print map
        System.out.println("Original map: " + map);
        System.out.println("Sorted map by key: " + sortedMapByKey);
        System.out.println("Sorted map by value: " + sortedMapByValue);
    }
}
```

# Sorting trong Java 8

## Sắp xếp các đối tượng bất kỳ

- Đôi khi ta cần sắp xếp 1 danh sách đối tượng bất kỳ chẳng hạn như đối tượng Student, Employee,… => cần định nghĩa 1 cách thức so sánh giữa các đối tượng để sắp xếp.
- Trong Java, với **Arrays.sort()** hoặc **Collections.sort()** có 2 cách để cung cấp **Comparator**:
  - Implement **Comparable** và override phương thức **compareTo(T obj)**
  - Implement **Comparator** và override phương thức **compare(T obj1, T obj2)**
- Giá trị trả về của 2 phương thức này:
  - Nếu **< 0**: giá trị ưu tiên của đối tượng thứ nhất lớn hơn đối tượng thứ hai. Khi thực hiện sắp xếp thì đối tượng thứ nhất sẽ đứng trước đối tượng thứ hai.
  - Nếu **= 0**: cả 2 có độ ưu tiên bằng nhau.
  - Nếu **> 0**: giá trị ưu tiên của đối tượng thứ nhất nhỏ hơn đối tượng thứ hai.

# Sorting trong Java 8

## Hiện thực Comparable và override phương thức compareTo(T obj)

```java
public class Student implements Comparable<Student> {

    private int id;
    private String name;
    private int age;

    public Student(int id, String name, int age) {

    @Override
    public int compareTo(Student s) {
        return this.getName().compareTo(s.getName());
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + ", age=" + age + "]";
    }
}
```

```java
public class SortedObjectExample01 {
    public static void main(String[] args) {

        List<Student> students = Arrays.asList( //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        );

        Collections.sort(students);
        students.forEach(System.out::println);
    }
}
```

Collections.sort()

```java
public class SortedObjectExample02 {
    public static void main(String[] args) {

        Student []students = { //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        };

        Arrays.sort(students);
        Stream.of(students).forEach(System.out::println);
    }
}
```

Arrays.sort()

# Sorting trong Java 8

## Hiện thực Comparator và override phương thức compare(T obj1, T obj2)

- Tạo **new class** cài đặt **Interface Comparator**

```
public class AgeComparator implements Comparator<Student> {

    @Override
    public int compare(Student s1, Student s2) {
        return s1.getAge() - s2.getAge();
    }

}
```

- Sử dụng Comparator với **Arrays.sort()**

```
public class SortedObjectExample04 {

    public static void main(String[] args) {

        Student []students = { //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        };

        Comparator<Student> nameComparator = new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
                return s1.getName().compareTo(s2.getName());
            }
        };

        Arrays.sort(students, nameComparator);
        Stream.of(students).forEach(System.out::println);
    }
}
```

- Sử dụng **Comparator** với **Collections.sort()**

```
public class SortedObjectExample03 {

    public static void main(String[] args) {

        List<Student> students = Arrays.asList( //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        );

        // new class comparator
        Comparator<Student> ageComparator = new AgeComparator();
        Collections.sort(students, ageComparator);

        // Anonymous function
        Comparator<Student> nameComparator1 = new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
                return s1.getName().compareTo(s2.getName());
            }
        };
        Collections.sort(students, nameComparator1);

        // Lambda
        Comparator<Student> nameComparator2 = (s1, s2) -> s1.getName().compareTo(s2.getName());
        Collections.sort(students, nameComparator2);

        students.forEach(System.out::println);
    }

    // Inner class
    class NameComparator implements Comparator<Student> {

        @Override
        public int compare(Student s1, Student s2) {
            return s1.getName().compareTo(s2.getName());
        }
    }
}
```

# Sorting trong Java 8

## Sắp xếp danh sách đối tượng sử dụng phương thức tham chiếu Java 8

- Ví dụ chúng ta có lớp **Helper**, lớp này chứa phương thức **compareByAge()**. Phương thức này hỗ trợ việc so sánh 2 đối tượng **Student** như bên dưới:

```
public class Helper {

    public static int compareByAge(Student s1, Student s2) {
        return s1.getAge() - s2.getAge();
    }
}
```

```
public class SortedObjectExample05 {
    public static void main(String[] args) {

        List<Student> students = Arrays.asList( //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        );

        Collections.sort(students, Helper::compareByAge);

        // Or
        students.sort(Helper::compareByAge);

        students.forEach(System.out::println);
    }
}
```

# Sorting trong Java 8

## Sắp xếp theo nhiều điều kiện sử dụng Comparator.comparing() và Comparator.thenComparing() trong Java 8

❑ Java 8 cung cấp 2 API mới hữu ích cho việc sắp xếp các phần tử là: **comparing()** và **thenComparing()** trong interface **Comparator**. Hai phương thức này khá thuận tiện cho việc sắp xếp chuỗi nhiều điều kiện của **Comparator**. Trong đó:

- ➢ **keyExtractor**: là một **Function**, có nhiệm vụ cung cấp thông tin khóa cần so sánh.
- ➢ **keyComparator**: là một **Comparator**, có nhiệm vụ cung cấp kết quả so sánh dựa trên khóa được cung cấp (keyExtractor). Tham số này không bắt buộc.

```
public static <T, U> Comparator<T> comparing(
            Function<? super T, ? extends U> keyExtractor) {}

public static <T, U> Comparator<T> comparing(
            Function<? super T, ? extends U> keyExtractor,
            Comparator<? super U> keyComparator) {}

default <U extends Comparable<? super U>> Comparator<T> thenComparing(
            Function<? super T, ? extends U> keyExtractor) {}

default <U> Comparator<T> thenComparing(
            Function<? super T, ? extends U> keyExtractor,
            Comparator<? super U> keyComparator) {}
```

# Sorting trong Java 8

## Sắp xếp theo nhiều điều kiện sử dụng Comparator.comparing() và Comparator.thenComparing() trong Java 8

- Ví dụ cần sắp xếp danh sách sinh viên theo tuổi, nếu cùng tuổi thì sắp xếp theo tên sử dụng phương thức **comparing()** và **thenComparing()** so với việc tự tạo **Comparator**.

```
public class SortedObjectExample06 {

    public static void main(String[] args) {

        List<Student> students = Arrays.asList( //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        );

        preJava8(students);
        java8(students);
    }

    private static void preJava8(List<Student> students) {

    private static void java8(List<Student> students) {
}
```

```
private static void preJava8(List<Student> students) {
    Comparator<Student> comparator = new Comparator<Student>() {
        @Override
        public int compare(Student s1, Student s2) {
            if (s1.getAge() == s2.getAge()) {
                return s1.getName().compareTo(s2.getName());
            }
            return s1.getAge() - s2.getAge();
        }
    };

    Collections.sort(students, comparator);
    System.out.println("PreJava8: ");
    students.forEach(System.out::println);
}

private static void java8(List<Student> students) {
    Comparator<Student> comparator = Comparator.comparing(Student::getAge)
                                               .thenComparing(Student::getName);

    Collections.sort(students, comparator);
    // Or
    students.sort(comparator);

    System.out.println("\nJava8: ");
    students.forEach(System.out::println);
}
```

# Sorting trong Java 8

## Sắp xếp đảo ngược (Reverse Order)

- Trước Java 8, chúng ta có thể sắp xếp đảo ngược một danh sách bằng cách đảo ngược bộ so sánh thông qua phương thức **Collections.reverseOrder( comparator)**.
- Với Java 8 cung cấp một phương thức khác giúp chúng ta có thể sắp xếp đảo ngược danh sách một cách trực tiếp thông qua phương thức **comparator.reversed()**.

```
public class SortedObjectExample07 {

    public static void main(String[] args) {

        List<Student> students = Arrays.asList( //
                new Student(1, "One", 22), //
                new Student(2, "Two", 18), //
                new Student(3, "Three", 20), //
                new Student(4, "Four", 19), //
                new Student(5, "Five", 22) //
        );

        // Anonymous function
        Comparator<Student> nameComparator = new Comparator<Student>() {
            @Override
            public int compare(Student s1, Student s2) {
                return s1.getName().compareTo(s2.getName());
            }
        };

        Collections.sort(students, Collections.reverseOrder(nameComparator));

        // Or
        Collections.sort(students, nameComparator.reversed());

        students.forEach(System.out::println);
    }
}
```

# Tổng kết nội dung bài học

- ❑ Optional trong Java 8
- ❑ Predicate trong Java 8
- ❑ String Joiner trong Java 8
- ❑ Sorting trong Java 8