VIETNAM AUSTRALIA Vocational College

# Slide-2.4: Collectors, Date Time, distinct(), removeIf()

*Giảng viên: Nguyễn Bá Minh Đạo*

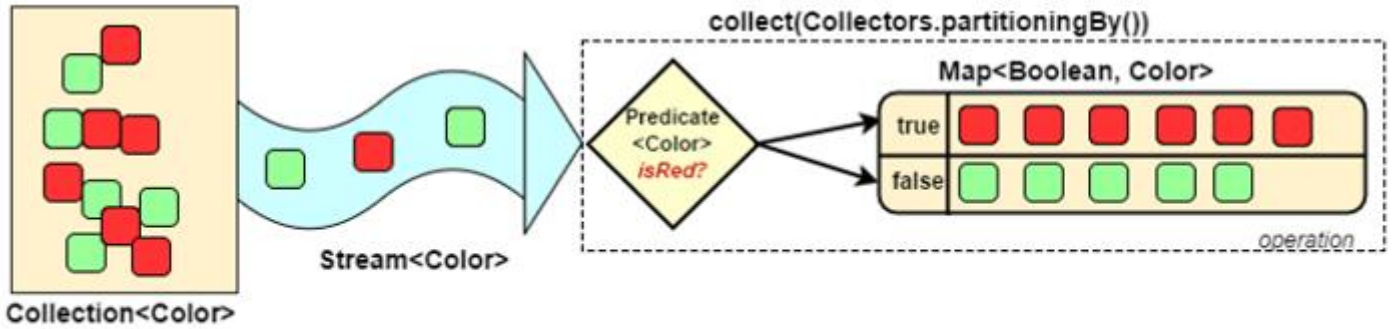
# ***Nội dung***

1. Collectors trong Java 8
2. Date Time trong Java 8
3. distinct(), removeIf()

# Collectors trong Java 8

## Giới thiệu lớp Collectors

- **Stream.collect()** là một trong các phương thức đầu cuối (terminal operation) của **Stream API** trong Java 8.
- **Stream.collect()** cho phép thực hiện các thao tác có thể thay đổi trên các phần tử được lưu giữ trong Stream.
- Ví dụ: chuyển các phần tử sang một số cấu trúc dữ liệu khác, áp dụng một số logic bổ sung, tính toán, …



# Collectors trong Java 8

## *Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API*

- **Collectors.toList()**: có thể sử dụng để thu thập các phần tử **Stream** vào một **List** (kết quả luôn là **ArrayList**)

```
List<String> result = list.stream().collect(Collectors.toList());
```

- **Collectors.toSet()**: có thể được sử dụng để thu thập tất cả các phần tử Stream vào một **Set** (kết quả luôn là **HashSet**).

```
List<String> result = list.stream().collect(Collectors.toSet());
```

- **Collectors.toCollection()**: Khi sử dụng **Collectors.toSet()** và **Collectors.toList()**, không thể xác định bất kỳ lớp cài đặt cụ thể nào của chúng như **LinkedList**, **LinkedHashSet**, **TreeSet**, … Nếu muốn sử dụng lớp cài đặt cụ thể, chúng ta cần phải sử dụng phương thức **Collectors.toCollection(c)** , với Collection được cung cấp tương ứng.

```
List<String> result = list.stream().collect(Collectors.toCollection(LinkedList::new));
```

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- **Collectors.toMap()**: được sử dụng để thu thập tất cả phần tử Stream vào một **Map**.

```
public static Collector<T, ?, M> toMap(Function<? super T, ? extends K> keyMapper,
                                       Function<? super T, ? extends U> valueMapper,
                                       BinaryOperator<U> mergeFunction,
                                       Supplier<M> mapSupplier) {}
```

- Trong đó:
  - **keyMapper**: một Function<T, R> để trích xuất một khóa (key) từ một phần tử Stream.
  - **valueMapper**: một Function<T, R> để trích xuất một giá trị (value) được liên kết với một khóa (key) đã cho.
  - **mergeFunction**: được sử dụng để giải quyết xung đột giữa các giá trị được liên kết với cùng 1 khóa. Tham số này không bắt buộc. Nhưng, nếu có **key** trùng sẽ báo lỗi.
  - **mapSupplier**: một **Supplier** trả về **instance** của **Map** mới, rỗng, kết quả được chèn vào. Tham số này không bắt buộc, mặc định là **HashMap**.

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- Ví dụ 1:

```
public class CollectorsExample01 {

    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "C++", "C#", "PHP");

        Map<String, Integer> result = list.stream()
                                          .collect(
                                                  Collectors.toMap(Function.identity(),
                                                                   String::length)
                                          );

        // => {C#=2, Java=4, C++=3, PHP=3}
    }
}
```

- **Function.identity()** : là một phương thức tiện ích để xác định đối số và kết quả trả về cùng một giá trị.

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- Ví dụ 2:

```
class Student {

    private String name;
    private Integer score;

    public Student(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public Integer getScore() {
        return score;
    }
}

public class CollectorsExample02 {
    public static void main(String[] args) {
}
```

```
public class CollectorsExample02 {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList( //
                new Student("B", 70), //
                new Student("A", 80), //
                new Student("C", 75), //
                new Student("A", 100) //
        );

        // Resolve collisions between values associated with the same key
        Map<String, Integer> result1 = students.stream().collect( //
                Collectors.toMap(Student::getName, Student::getScore, //
                        (s1, s2) -> (s1 > s2 ? s1 : s2) // BinaryOperator
                ));
        System.out.println("result1 : " + result1);

        // Identify the instance of LinkedHashMap
        Map<String, Integer> result2 = students.stream().collect( //
                Collectors.toMap(Student::getName, Student::getScore, //
                        (s1, s2) -> (s1 > s2 ? s1 : s2), // BinaryOperator
                        LinkedHashMap::new // Supplier
                ));
        System.out.println("result2 : " + result2);
    }
}
```

# Collectors trong Java 8

## *Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API*

- **Collectors.collectingAndThen()**: là một bộ thu thập đặc biệt cho phép thực hiện một hành động khác ngay lập tức sau khi thu thập kết thúc.

```
List<String> list = Arrays.asList("Java", "C++", "C#", "PHP");

List<String> result5 = list.stream().collect(
            Collectors.collectingAndThen(Collectors.toList(), x -> x.subList(0, 2))
);

// => [Java, C++]
```

- **Collectors.counting()**: được sử dụng để đếm các phần tử trong **Stream**.

```
List<String> list = Arrays.asList("Java", "C++", "C#", "PHP");

Long result9 = list.stream().collect(Collectors.counting()); // => 4
```

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- **Collectors.joining()**: có thể được sử dụng join các phần tử Stream<String>

```
public static Collector<CharSequence, ?,
String> joining(CharSequence delimiter,
CharSequence prefix,
CharSequence suffix) {}
```

- Trong đó:
  - **delimiter**: chuỗi ký tự phần tách các phần tử.
  - **prefix**: chuỗi ký tự được thêm vào đầu kết quả.
  - **suffix**: chuỗi ký tự được thêm vào cuối kết quả.

- Ví dụ:

```
List<String> list = Arrays.asList("Java", "C++", "C#", "PHP");

String result6 = list.stream().collect(Collectors.joining()); // => JavaC++C#PHP

String result7 = list.stream().collect(Collectors.joining(", ")); // => Java, C++, C#, PHP

String result8 = list.stream().collect(Collectors.joining(" ", "PRE-", "-POST")); // => PRE-Java C++ C# PHP-POST
```

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- **Collectors.summarizingDouble/Long/Int()**: trả về một lớp đặc biệt chứa thông tin thống kê về dữ liệu số trong Stream các phần tử được trích xuất.

```
List<String> list = Arrays.asList("Java", "C++", "C#", "PHP");
IntSummaryStatistics result10 = list.stream().collect(Collectors.summarizingInt(String::length));
// => IntSummaryStatistics{count=4, sum=12, min=2, average=3.000000, max=4}
```

- **Collectors.averagingDouble/Long/Int()**: trả về giá trị trung bình của các phần tử.

```
Double result = list.stream().collect(Collectors.averagingDouble(String::length));
```

- **Collectors.summingDouble/Long/Int()**: trả về giá trị tổng của các phần tử.

```
Double result = list.stream().collect(Collectors.summingDouble(String::length));
```

- **Collectors.maxBy()/ Collectors.minBy()**: trả về giá trị lớn nhất/ nhỏ nhất của một Stream theo bộ Comparator được cung cấp.

```
Optional<String> result = list.stream().collect(Collectors.maxBy(Comparator.naturalOrder())); // => PHP
Optional<String> result = list.stream().collect(Collectors.minBy(Comparator.naturalOrder())); // => C#
```

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- **Collectors.groupingBy()**: được sử dụng để nhóm các đối tượng theo một số thuộc tính và lưu trữ các kết quả trong một **Map**.

```
class Book {
    private Integer id;
    private String title;
    private Integer categoryId;

    public Book(Integer id, String title, Integer categoryId) {
        super();
        this.id = id;
        this.title = title;
        this.categoryId = categoryId;
    }

    public Integer getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public Integer getCategoryId() {
        return categoryId;
    }

    @Override
    public String toString() {
        return "Book [id=" + id + ", title=" + title + ", "
                           + "categoryId=" + categoryId + "]";
    }

}
```

```
public class CollectorsExample03 {
    public static void main(String[] args) {
        List<Book> books = Arrays.asList( //
                new Book(1, "A", 1), //
                new Book(2, "B", 1), //
                new Book(3, "C", 2), //
                new Book(4, "D", 3), //
                new Book(5, "E", 1) //
        );

        Map<Integer, Set<Book>> result = books.stream()
                .collect(Collectors.groupingBy(
                        Book::getCategoryId, Collectors.toSet())
                );
        result.forEach((catId, booksInCat)
                        -> System.out.println("Category " + catId
                                                + " : "
                                                + booksInCat.size()));
    }
}
```

# Collectors trong Java 8

## *Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API*

- **Collectors.partitioningBy()**: là một trường hợp đặc biệt của **Collectors.groupingBy()** chấp nhận một **Predicate** và thu thập các phần tử của **Stream** vào một **Map** với các giá trị **Boolean** như **khóa** (key) và **Collection** như **giá trị** (value).
  - key = true: là một tập hợp các phần tử phù hợp với **Predicate** đã cho.
  - key = false: là một tập hợp các phần tử không khớp với **Predicate** đã cho.

```
public class CollectorsExample04 {

    public static void main(String[] args) {
        List<Book> books = new ArrayList<Book>();
        books.add(new Book(1, "A", 1));
        books.add(new Book(2, "B", 1));
        books.add(new Book(3, "C", 2));
        books.add(new Book(4, "D", 3));
        books.add(new Book(5, "E", 1));

        Map<Boolean, Set<Book>> partitioningBy = books.stream()
                .collect(Collectors.partitioningBy(b -> b.getCategoryId() > 2, Collectors.toSet()));
            System.out.println(partitioningBy);
    }
}
```

# Collectors trong Java 8

## *Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API*

- **Collectors.reducing()**: thực hiện giảm các phần tử đầu vào của nó trong một **BinaryOperator** được chỉ định.
- Cú pháp:

```
public static <T> Collector<T,?,T> reducing(T identity, BinaryOperator<T> op) {}

public static <T> Collector<T, ?, Optional<T>> reducing(BinaryOperator<T> op) {}
```

- Trong đó:
  - **identity**: giá trị khởi tạo để thực hiện reduction (cũng là giá trị được trả về khi không có phần tử đầu vào)
  - **op**: một **BinaryOperator<T>** được sử dụng để giảm các phần tử đầu vào.

# Collectors trong Java 8

## Lớp Collectors cung cấp nhiều phương thức xử lý các phần tử Stream API

- Ví dụ sử dụng **reducing()** để tìm nhân viên lớn tuổi nhất theo từng company, tính tổng lương phải trả cho tất cả nhân viên.

```java
public class Employee {
    private String name;
    private Integer age;
    private String companyName;
    private Integer salary;

    public Employee(String name, Integer age,
            String companyName, Integer salary) {
        this.name = name;
        this.age = age;
        this.companyName = companyName;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public Integer getAge() {
        return age;
    }

    public String getCompanyName() {
        return companyName;
    }

    public Integer getSalary() {
        return salary;
    }
}
```

```java
public class CollectorsExample05 {
    public static void main(String[] args) {

        List<Employee> list = Arrays.asList( //
                new Employee("Emp1", 22, "A", 50), //
                new Employee("Emp2", 23, "A", 60), //
                new Employee("Emp3", 22, "B", 40), //
                new Employee("Emp4", 21, "B", 70) //
        );

        // Find employees with the maximum age of each company
        Comparator<Employee> ageComparator = Comparator.comparing(Employee::getAge);

        Map<String, Optional<Employee>> map = list.stream().collect(
                Collectors.groupingBy(Employee::getCompanyName,
                        Collectors.reducing(BinaryOperator.maxBy(ageComparator))));

        map.forEach((k, v) -> System.out.println(
                "Company: " + k +
                ", Age: " + ((Optional<Employee>) v).get().getAge() +
                ", Name: " + ((Optional<Employee>) v).get().getName()));

        // Summary salary
        Integer bonus = 30;
        Integer totalSalaryExpense = list.stream()
                .map(emp -> emp.getSalary())
                .reduce(bonus, (a, b) -> a + b);
        System.out.println("Total salary expense: " + totalSalaryExpense);
    }
}
```

# Date Time trong Java 8

## Tại sao chúng ta cần API Date Time mới?

- Các class **Java Date Time** không nhất quán: chúng ta có class Date trong cả 2 package **java.util** và **java.sql** . Trong khi class **format/ parse** chúng lại nằm ở trong package **java.text**.

- **java.util.Date** bao gồm cả **date và time** trong khi **java.sql.Date** chỉ bao gồm **date**. Cả 2 class đều có cùng tên, đây là một thiết kế khá xấu.

# Date Time trong Java 8

## Tại sao chúng ta cần API Date Time mới?

- Không rõ ràng trong việc định nghĩa các class cho time, timestamp, formatting và parsing.

- Chúng ta có lớp trừu tượng **java.text.DateFormat** để parse và format Date Time cần. Lớp **SimpleDateFormat** thường được sử dụng nhất.

# Date Time trong Java 8

## Tại sao chúng ta cần API Date Time mới?

- Tất cả các class Date đều **mutable**,do đó chúng không an toàn khi dùng chung trong môi trường đa luồng (Multi-Thread). Đây có thể coi là 1 vấn đề lớn với các class Java Date và Calendar.

- Class Date không cung cấp **Internationalization** và không hỗ trợ **Timezone**, sau đó các class **java.util.Calendar** và **java.util.TimeZone** được giới thiệu nhưng chúng cũng gặp phải các vấn đề bên trên.

# Date Time trong Java 8

## Những thay đổi về Date & Time trong Java 8

- **API Date Time trong Java 8** cài đặt các phương thức theo chuẩn **JSR-310**.
- **API Date Time trong Java 8** được thiết kế để khắc phục tất cả các sai sót trong việc triển khai **Date Time** trước đó.

# Date Time trong Java 8

## Những thay đổi về Date & Time trong Java 8

❑ **Một số nguyên tắc thiết kế** của **API Date Time** mới là:

➢ Khả năng immutable (**Immutability**): Tất cả các class trong **Date Time API** mới đều là immutable, điều này tốt cho môi trường đa luồng (Multi-Thread)

➢ Phân tách các mối quan tâm (**Separation of Concerns**): API mới phân chia rõ ràng giữa Date Time, hạn chế nhầm lẫn khi sử dụng. Nó định nghĩa phân biệt các class cho **Date**, **Time**, **DateTime**, **Timestamp**, **Timezone**, …

# Date Time trong Java 8

## Những thay đổi về Date & Time trong Java 8

❑ **Một số nguyên tắc thiết kế** của **API Date Time** mới là:

➢ Tính trong suốt (**Clarity**): Các method được định nghĩa rõ ràng và thực thi các hành động giống nhau trong cùng 1 class. Ví dụ, để lấy thời gian hiện tại ta có method **now()**, các method **format()** và **parse()** đều được định nghĩa trong những class này. Tất cả các lớp đều sử dụng **Factory Pattern** và **Strategy Pattern** để xử lý tốt hơn.

# Date Time trong Java 8

## Những thay đổi về Date & Time trong Java 8

❑ **Một số nguyên tắc thiết kế** của **API Date Time** mới là:

➢ Các tính năng hữu ích (**Utility operations**): Tất cả các class trong **API Date Time** đi kèm với các tác vụ phổ biến như cộng, trừ, parsing, định dạng date/time…

➢ Khả năng mở rộng (**Extendable**): **API Date Time** mới làm việc theo chuẩn **ISO-8601** calendar, nhưng ta vẫn có thể sử dụng nó với các chuẩn không phải ISO.

# Date Time trong Java 8

## Các package của API Date Time trong Java 8

- **java.time**: đây là 1 package chính của API Date Time. Bao gồm các class: LocalDate, LocalTime, LocalDateTime, Instant, Period, Duration,…Tất cả các class này đều là **immutable** và **thread safe**.
- **java.time.chrono**: package gồm các **generic API** được định nghĩa cho các hệ thống có lịch không theo chuẩn **ISO**. Chúng ta có thể mở rộng class **AbstractChronology** để tạo **Calendar** riêng cho hệ thống của mình.

# Date Time trong Java 8

## Các package của API Date Time trong Java 8

- **java.time.format** : **package** này gồm các **class** sử dụng cho **formatting** và **parsing**.
- **java.time.temporal** : **package** này gồm các **temporal object** và chúng ta có thể dùng chúng để lấy ngày cụ thể hoặc thời gian liên quan. Ví dụ, chúng ta có thể sử dụng chúng để tìm ra ngày đầu tiên hoặc ngày cuối cùng của tháng. Bạn có thể xác định các phương thức này một cách dễ dàng vì chúng luôn có định dạng **withXXX()**
- **java.time.zone***: **package** này gồm các **class** hỗ trợ các **Time zone** khác nhau.

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **LocalDate**: Tạo 1 instance của date theo tiêu chuẩn ISO 8601, không Time/Time Zone.

```
public class LocalDateExample {

    public static void main(String[] args) {

        // Current Date
        LocalDate today = LocalDate.now();
        System.out.println("Current Date = " + today);

        // Creating LocalDate by providing input arguments
        LocalDate firstDay_2014 = LocalDate.of(2014, Month.JANUARY, 1);
        System.out.println("Specific Date = " + firstDay_2014);

        // Try creating date by providing invalid inputs
        // LocalDate feb29_2014 = LocalDate.of(2014, Month.FEBRUARY, 29);
        // Exception in thread "main" java.time.DateTimeException:
        // Invalid date 'February 29' as '2014' is not a leap year

        // Current date in "Asia/Ho_Chi_Minh", you can get it from ZoneId javadoc
        LocalDate todayHCM = LocalDate.now(ZoneId.of("Asia/Ho_Chi_Minh"));
        System.out.println("Current Date in IST = " + todayHCM);

        // java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        // LocalDate todayIST = LocalDate.now(ZoneId.of("IST"));

        // Getting date from the base date i.e 01/01/1970
        LocalDate dateFromBase = LocalDate.ofEpochDay(365);
        System.out.println("365th day from base date = " + dateFromBase);

        // Obtains an instance of LocalDate from a year and day-of-year
        LocalDate hundredDay2014 = LocalDate.ofYearDay(2014, 100);
        System.out.println("100th day of 2014 = " + hundredDay2014);
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **LocalTime**: Tạo 1 instance của time theo tiêu chuẩn ISO 8601, không Date/Time Zone.

```
public class LocalTimeExample {

    public static void main(String[] args) {

        // Current Time
        LocalTime time = LocalTime.now();
        System.out.println("Current Time=" + time);

        // Creating LocalTime by providing input arguments
        // LocalTime.of(int hour, int minute, int second, int nanoOfSecond)
        LocalTime specificTime = LocalTime.of(12, 20, 25, 40);
        System.out.println("Specific Time of Day = " + specificTime);

        // Try creating time by providing invalid inputs
        // LocalTime invalidTime = LocalTime.of(25,20);
        // Exception in thread "main" java.time.DateTimeException:
        // Invalid value for HourOfDay (valid values 0 - 23): 25

        // Current date in "Asia/Ho_Chi_Minh", you can get it from ZoneId javadoc
        LocalTime timeHCM = LocalTime.now(ZoneId.of("Asia/Ho_Chi_Minh"));
        System.out.println("Current Time in IST = " + timeHCM);

        // java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        // LocalTime todayIST = LocalTime.now(ZoneId.of("IST"));

        // Getting date from the base date i.e 01/01/1970
        LocalTime specificSecondTime = LocalTime.ofSecondOfDay(10000);
        System.out.println("10000th second time = " + specificSecondTime);

    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **LocalDateTime**: bao gồm cả 2 API trên, tạo ra instance chứa cả Date, Time và không có Time Zone

```java
public class LocalDateTimeExample {

    public static void main(String[] args) {

        // Current Date
        LocalDateTime today = LocalDateTime.now();
        System.out.println("Current DateTime = " + today);

        // Current Date using LocalDate and LocalTime
        today = LocalDateTime.of(LocalDate.now(), LocalTime.now());
        System.out.println("Current DateTime = " + today);

        // Creating LocalDateTime by providing input arguments
        // LocalDateTime.of(int year, Month month, int dayOfMonth, int hour, int minute, int second)
        LocalDateTime specificDate = LocalDateTime.of(2014, Month.JANUARY, 1, 10, 10, 30);
        System.out.println("Specific Date = " + specificDate);

        // Try creating date by providing invalid inputs
        // LocalDateTime feb29_2014 = LocalDateTime.of(2014, Month.FEBRUARY, 28, 25,1,1);
        // Exception in thread "main" java.time.DateTimeException:
        // Invalid value for HourOfDay (valid values 0 - 23): 25

        // Current date in "Asia/Ho_Chi_Minh", you can get it from ZoneId javadoc
        LocalDateTime todayHCM = LocalDateTime.now(ZoneId.of("Asia/Ho_Chi_Minh"));
        System.out.println("Current Date in IST = " + todayHCM);

        // java.time.zone.ZoneRulesException: Unknown time-zone ID: IST
        // LocalDateTime todayIST = LocalDateTime.now(ZoneId.of("IST"));

        // Getting date from the base date i.e 01/01/1970
        LocalDateTime dateFromBase = LocalDateTime.ofEpochSecond(10000, 0, ZoneOffset.UTC);
        System.out.println("10000th second time from 01/01/1970 = " + dateFromBase);

    }

}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **ZonedDateTime**: bao gồm cả API LocalDateTime có Time Zone.

```
public class ZonedDateTimeExample {

    public static void main(String[] args) {

        // Get all available zones
        Set<String> allZoneIds = ZoneId.getAvailableZoneIds();
        System.out.println("allZoneIds = " + allZoneIds);

        ZoneId zoneHCM = ZoneId.of("Asia/Ho_Chi_Minh");

        // Creating LocalDateTime by providing input arguments
        LocalDateTime today = LocalDateTime.now();
        System.out.println("LocalDateTime = " + today);

        // Creating ZonedDateTime by providing input arguments
        ZonedDateTime hcmDateTime = ZonedDateTime.of(today, zoneHCM);
        System.out.println("ZonedDateTime = " + hcmDateTime);

        // using offsets
        ZoneOffset offset = ZoneOffset.of("+05:00");
        System.out.println("offset = " + offset);

        OffsetDateTime todayPlusFive = OffsetDateTime.of(today, offset);
        System.out.println("todayPlusFive = " + todayPlusFive);

        OffsetDateTime todayMinusTwo = todayPlusFive.withOffsetSameInstant(ZoneOffset.ofHours(-2));
        System.out.println("todayMinusTwo = " + todayMinusTwo);
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **Instant**: hỗ trợ làm việc với **timestamps**, sử dụng để ghi lại thời gian sự kiện trong ứng dụng. Nó lưu trữ dưới định dạng của **unix timestamp**. Ví dụ: 1970-01-01T00:00:00Z.

```
public class InstantExample {

    public static void main(String[] args) {

        // Current timestamp
        Instant now = Instant.now();
        System.out.println("Current Timestamp = " + now);

        // Instant from timestamp
        Instant specificTime = Instant.ofEpochMilli(now.toEpochMilli());
        System.out.println("Specific Time = " + specificTime);

        // Obtain an instance of Instant from a text string
        Instant specifyString = Instant.parse("2018-06-20T10:37:30.00Z");
        System.out.println("specifyString = " + specifyString);

        // Obtains a Duration representing a number of standard 24 hour days
        // return Duration with format of days*24
        Duration thirtyDay = Duration.ofDays(30);
        System.out.println(thirtyDay);

        // Copy of this instant with the specified amount subtracted
        Instant minus5 = now.minus(Duration.ofDays(5));
        System.out.println("minus5 = " + minus5);

        // Copy of this instant with the specified amount added
        Instant plus5 = now.plus(Duration.ofDays(5));
        System.out.println("plus5 = " + plus5);
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **Period**: hỗ trợ tính toán năm/tháng/ngày giữa 2 Date Time

```
public class PeriodExample {

    public static void main(String[] args) {

        LocalDate firstDate = LocalDate.now();
        LocalDate secondDate = LocalDate.of(2017, 5, 20);
        System.out.println("firstDate: " + firstDate); // 2018-06-23
        System.out.println("secondDate: " + secondDate); // 2017-05-20

        Period period = Period.between(firstDate, secondDate);
        System.out.println("period: " + period); // P-1M-3D

        int days = period.getDays();
        int months = period.getMonths();
        int years = period.getYears();
        boolean isNegative = period.isNegative();
        System.out.println("days: " + days); // -3
        System.out.println("months: " + months); // -1
        System.out.println("years: " + years); // -1
        System.out.println("isNegative: " + isNegative); // true

        Period twoMonthTenDays = Period.ofMonths(2).plusDays(10);
        System.out.println("twoMonthTenDays: " + twoMonthTenDays); // P2M10D

        LocalDate plusDate = firstDate.plus(twoMonthTenDays);
        System.out.println("plusDate: " + plusDate); // 2018-09-02

        LocalDate minusDate = firstDate.minus(twoMonthTenDays);
        System.out.println("minusDate: " + minusDate); // 2018-04-13
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

❑ **Duration**: hỗ trợ tính toán chi tiết đến seconds and nanoseconds giữa 2 Date Time.

```java
public class DurationExample {

    public static void main(String[] args) {

        LocalDateTime firstDate = LocalDateTime.now();
        LocalDateTime secondDate = LocalDateTime.of(2018, 6, 20, 0, 0, 0);
        System.out.println("firstDate: " + firstDate); // 2018-06-23T21:31:28.924
        System.out.println("secondDate: " + secondDate); // 2018-06-20T00:00

        // Obtains a Duration representing the duration between two temporal objects
        // The temporal objects are Instant or LocalDateTime
        Duration duration = Duration.between(firstDate, secondDate);
        System.out.println("duration: " + duration); // PT-93H-31M-28.924S

        long days = duration.toDays();
        long hours = duration.toHours();
        long minutes = duration.toMinutes();
        long seconds = duration.getSeconds();
        long millis = duration.toMillis();
        long nanos = duration.toNanos();
        System.out.println("days: " + days); // -3
        System.out.println("hours: " + hours); // -93
        System.out.println("minutes: " + minutes); // -5611
        System.out.println("seconds: " + seconds); // -336689
        System.out.println("millis: " + millis); // -336688924
        System.out.println("nanos: " + nanos); // -336688924000000

        Duration twoHours = Duration.ofHours(2);
        System.out.println("twoHours: " + twoHours); // PT2H

        LocalDateTime plusDate = firstDate.plus(twoHours);
        System.out.println("plusDate: " + plusDate); // 2018-06-23T23:35:21.045

        LocalDateTime minusDate = firstDate.minus(twoHours);
        System.out.println("minusDate: " + minusDate); // 2018-06-23T19:35:21.045
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **Year, YearMonth**: **Year** sẽ tạo ra 1 **instance** của **date** theo tiêu chuẩn **ISO 8601**, chỉ có **Year** (Year) hoặc **Year và Month** (YearMonth).

```
public class YearMonthExample {

    public static void main(String[] args) {

        // Year Example

        Year currentYear = Year.now();
        System.out.println("currentYear: " + currentYear); // 2018

        Year specifyYear = Year.of(2016);
        System.out.println("specifyYear: " + specifyYear); // 2016
        System.out.println("isLeap : " + specifyYear.isLeap()); // true

        int dayOfYear = 100;
        LocalDate localDate = currentYear.atDay(dayOfYear);
        System.out.println("localDate: " + localDate); // 2018-04-10

        // YearMonth Example

        YearMonth currentYearMonth = YearMonth.now();
        System.out.println("currentYearMonth: " + currentYearMonth);

        YearMonth specifyYearMonth = YearMonth.of(2016, 1);
        System.out.println("specifyYearMonth: " + specifyYearMonth);

        int dayOfMonth = 20;
        LocalDate localDate2 = currentYearMonth.atDay(dayOfMonth);
        System.out.println("localDate2: " + localDate2); // 2018-06-20

        // Year -> YearMonth

        YearMonth ym = currentYear.atMonth(5);
        System.out.println("ym: " + ym); // 2018-05
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **DayOfWeek, Month Enum**: các enum thể hiện các ngày trong tuần, tháng. Hỗ trợ một số thao tác tính toán, biểu diễn ngày trong tuần, tháng.

```java
public class WeekMonthEnumExample {

    public static void main(String[] args) {

        // DayOfWeek Enum Example
        DayOfWeek monday = DayOfWeek.MONDAY;
        System.out.println(monday); // MONDAY
        System.out.println(monday.getDisplayName(TextStyle.SHORT, Locale.getDefault())); // Mon
        System.out.println(monday.getDisplayName(TextStyle.FULL, Locale.getDefault())); // Monday
        System.out.println(monday.plus(5)); // SATURDAY
        System.out.println(DayOfWeek.of(1)); // MONDAY
        System.out.println(DayOfWeek.of(7)); // SUNDAY
        System.out.println(DayOfWeek.valueOf("SUNDAY")); // SUNDAY
        System.out.println(monday.compareTo(DayOfWeek.SUNDAY)); // -6

        // Month Enum Example
        Month april = Month.APRIL;
        System.out.println(april); // APRIL
        System.out.println(april.getDisplayName(TextStyle.SHORT, Locale.getDefault())); // Apr
        System.out.println(april.getDisplayName(TextStyle.FULL, Locale.getDefault())); // April
        System.out.println(april.plus(3)); // JULY
        System.out.println(Month.FEBRUARY.maxLength()); // 29
        System.out.println(Month.FEBRUARY.minLength()); // 28
        System.out.println(april.firstDayOfYear(true)); // 92
        System.out.println(Month.of(1)); // JANUARY
        System.out.println(Month.of(12)); // DECEMBER
        System.out.println(Month.valueOf("FEBRUARY")); // FEBRUARY
        System.out.println(april.compareTo(Month.FEBRUARY)); // 2
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

❑ **ChronoUnit, ChronoField Enum**:

➢ **ChronoUnit Enum**: được sử dụng để đo lượng/tính toán thời gian như: năm, tháng, ngày, giờ, phút, giây. Ví dụ: YEAR, MONTHS, WEEKS, HOURS,…

➢ **ChronoField Enum**: được sử dụng để truy xuất một phần phần nào đó của **Date Time**. Ví dụ: DAY_OF_MONTH, DAY_OF_WEEK, MONTH_OF_YEAR, YEAR, …

```
public class ChronoFieldUnitExample {

    public static void main(String[] args) {

        // ====== ChronoUnit Example ======

        // Get the current date
        LocalDate today = LocalDate.now();
        System.out.println("Current date: " + today);

        // add 1 week to the current date
        LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);
        System.out.println("Next week: " + nextWeek);

        // add 1 week to the current date
        LocalDate previousWeek = today.minus(1, ChronoUnit.WEEKS);
        System.out.println("Previous week: " + previousWeek);

        // add 1 month to the current date
        LocalDate nextMonth = today.plus(1, ChronoUnit.MONTHS);
        System.out.println("Next month: " + nextMonth);

        // add 1 year to the current date
        LocalDate nextYear = today.plus(1, ChronoUnit.YEARS);
        System.out.println("Next year: " + nextYear);

        // add 10 years to the current date
        LocalDate nextDecade = today.plus(1, ChronoUnit.DECADES);
        System.out.println("Date after ten year: " + nextDecade);

        // ====== ChronoField Example ======

        LocalDateTime currentDateTime = LocalDateTime.now();
        System.out.println("\ncurrentDateTime: " +currentDateTime);
        System.out.println("Year: " + currentDateTime.get(ChronoField.YEAR));
        System.out.println("Month: " + currentDateTime.get(ChronoField.MONTH_OF_YEAR));
        System.out.println("Day of month: " + currentDateTime.get(ChronoField.DAY_OF_MONTH));
        System.out.println("Hour of day: " + currentDateTime.get(ChronoField.HOUR_OF_DAY));
        System.out.println("Minute of hour: " + currentDateTime.get(ChronoField.MINUTE_OF_HOUR));
    }
}
```

# Date Time trong Java 8

## Các ví dụ với API Date Time trong Java 8

- **TemporalAdjuster**: có thể được sử dụng để điều chỉnh ngày và tính toán, so sánh khoảng thời gian giữa hai ngày.

```
public class DateApiUtilities {

    public static void main(String[] args) {

        LocalDate today = LocalDate.now();

        // Get the Year, check if it's leap year
        System.out.println("Year " + today.getYear() + " is Leap Year? " + today.isLeapYear());

        // Compare two LocalDate for before and after
        System.out.println("Today is before 01/01/2018? " + today.isBefore(LocalDate.of(2018, 1, 1)));

        // Create LocalDateTime from LocalDate
        System.out.println("Current Time = " + today.atTime(LocalTime.now()));

        // plus and minus operations
        System.out.println("10 days after today will be " + today.plusDays(10));
        System.out.println("3 weeks after today will be " + today.plusWeeks(3));
        System.out.println("20 months after today will be " + today.plusMonths(20));

        System.out.println("10 days before today will be " + today.minusDays(10));
        System.out.println("3 weeks before today will be " + today.minusWeeks(3));
        System.out.println("20 months before today will be " + today.minusMonths(20));

        // Temporal adjusters for adjusting the dates
        LocalDate firstDayOfThisMonth = today.with(TemporalAdjusters.firstDayOfMonth());
        System.out.println("First date of this month= " + firstDayOfThisMonth);

        LocalDate lastDayOfThisMonth = today.with(TemporalAdjusters.lastDayOfMonth());
        System.out.println("Last date of this month= " + lastDayOfThisMonth);

        LocalDate lastDayOfYear = today.with(TemporalAdjusters.lastDayOfYear());
        System.out.println("Last date of this year= " + lastDayOfYear);

        Period period = today.until(lastDayOfYear);
        System.out.println("Period Format= " + period);
        System.out.println("Months remaining in the year= " + period.getMonths());
    }
}
```

# Date Time trong Java 8

## Định dạng (Format), phân tích (Parse) Date Time trong Java 8

- **format(pattern)**: được sử dụng để định dạng một giá trị **Date Time** sang chuỗi tương ứng với **pattern** được cung cấp.
- **parse(str, pattern)**: được sử dụng để phân tích chuỗi bất kỳ với **pattern** được cung cấp sang kiểu **Date Time** tương ứng.

```
public class DateParseFormatExample {

    public static void main(String[] args) {

        // Format LocalDate examples
        LocalDate date = LocalDate.now();
        System.out.println("Default format of LocalDate = " + date);
        System.out.println(date.format(DateTimeFormatter.ofPattern("d::MMM::uuuu")));
        System.out.println(date.format(DateTimeFormatter.BASIC_ISO_DATE));

        // Format LocalDateTime examples
        LocalDateTime dateTime = LocalDateTime.now();
        System.out.println("\nDefault format of LocalDateTime = " + dateTime);
        System.out.println(dateTime.format(DateTimeFormatter.ofPattern("d::MMM::uuuu HH::mm::ss")));
        System.out.println(dateTime.format(DateTimeFormatter.BASIC_ISO_DATE));

        // Format Instant Example
        Instant timestamp = Instant.now();
        System.out.println("\nDefault format of Instant = " + timestamp);

        // Parse examples
        LocalDateTime dt = LocalDateTime.parse("27::Apr::2014 21::39::48",
                DateTimeFormatter.ofPattern("d::MMM::uuuu HH::mm::ss"));
        System.out.println("\nDefault format after parsing = " + dt);
    }
}
```

# Date Time trong Java 8

## Chuyển đổi qua lại giữa các kiểu dữ liệu Date Time trong Java 8

```
public class DateTimeConversionExample {

    public static void main(String[] args) {

        // LocalDate/ LocalTime <-> LocalDateTime/ ZonedDateTime
        LocalDate date = LocalDate.now();
        LocalTime time = LocalTime.now();
        LocalDateTime dateTimeFromDateAndTime = LocalDateTime.of(date, time);
        LocalDate dateFromDateTime = dateTimeFromDateAndTime.toLocalDate();
        LocalTime timeFromDateTime = dateTimeFromDateAndTime.toLocalTime();
        ZonedDateTime hcmDateTime = ZonedDateTime.of(dateTimeFromDateAndTime, ZoneId.of("Asia/Ho_Chi_Minh"));

        // Convert old classes to Java 8 Date Time
        Instant instantFromDate = new Date().toInstant();
        ZoneId zoneId = TimeZone.getDefault().toZoneId();
        Instant instantFromCalendar = Calendar.getInstance().toInstant();
        ZonedDateTime zonedDateTime = new GregorianCalendar().toZonedDateTime();

        // Instant <-> LocalDateTime
        Instant instant = Instant.now();
        LocalDateTime dateTimeFromInstant = LocalDateTime.ofInstant(instant, ZoneId.systemDefault());
        Instant instantFromLocalDateTime = dateTimeFromInstant.toInstant(ZoneOffset.ofHours(+7));

        // Instant <-> LocalDate
        LocalDate localDate = instant.atZone(ZoneId.systemDefault()).toLocalDate();
        Instant instantFromLocalDate = localDate.atStartOfDay(ZoneId.systemDefault()).toInstant();

        // Convert Java 8 Date Time to old classes
        Date dateFromInstant = Date.from(Instant.now());
        TimeZone timeZone = TimeZone.getTimeZone(ZoneId.of("Asia/Ho_Chi_Minh"));
        GregorianCalendar gregorianCalendar = GregorianCalendar.from(ZonedDateTime.now());
    }
}
```

# Remove Duplication trong Java 8

## Sử dụng phương thức distinct() trong Stream API

- Phương thức **distinct()** trả về một **Stream** gồm các phần tử duy nhất, việc xác định các phần tử trùng lặp được so sánh theo phương thức **Object.equals(Object)**

```
public class RemoveDuplicateInArrayList01 {

    public static void main(String[] args) {
        List<String> listWithDuplicateElements = new ArrayList<String>();
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("J2EE");
        listWithDuplicateElements.add("JSP");
        listWithDuplicateElements.add("SERVLETS");
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("STRUTS");
        listWithDuplicateElements.add("JSP");

        List<String> listWithoutDuplicateElements = listWithDuplicateElements
                .stream()
                .distinct()
                .collect(Collectors.toList());
        System.out.println(listWithoutDuplicateElements); // [JAVA, J2EE, JSP, SERVLETS, STRUTS]
    }
}
```

# Remove Duplication trong Java 8

## Sử dụng phương thức removeIf() trong Stream API

- Phương thức **removeIf()** chấp nhận đối số là 1 **Predicate**, nó loại bỏ tất cả các phần tử của **Collection** thỏa mãn điều kiện đã cho.

```
public class RemoveDuplicateInArrayList02 {

    public static void main(String[] args) {
        List<String> listWithDuplicateElements = new ArrayList<String>();
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("J2EE");
        listWithDuplicateElements.add("JSP");
        listWithDuplicateElements.add("SERVLETS");
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("STRUTS");
        listWithDuplicateElements.add("JSP");

        Set<String> elementsAlreadySeen = new LinkedHashSet<>();
        listWithDuplicateElements.removeIf(s -> !elementsAlreadySeen.add(s));
        System.out.println(elementsAlreadySeen); // [JAVA, J2EE, JSP, SERVLETS, STRUTS]
    }
}
```

# Tổng kết nội dung bài học

- Collectors trong Java 8
- Date Time trong Java 8
- distinct(), removeIf()