**Fitur Java 8 Yang Dipakai**

1. **Lambda Expressions**

* Lambda expressions menjadi fitur terbesar yang dikembangkan di Java versi 8.
* Lambda expressions memfasilitasi functional programming dan sangat mempermudah proses development.
* Karakteristik syntax Lambda expressions:

|  |
| --- |
| parameter -> expression body |

* Contoh penggunaan Lambda Expressions pada project template kita yang baru, ada pada file service implementation AppUserServiceImpl.java

|  |
| --- |
| @Override  public AppUser findById(Long id) throws ResourceNotFoundException {  return appUserMapper.findById(id)  .orElseThrow(**() -> new ResourceNotFoundException(AppUser.class, id)**);  } |

***\*notes:*** *Penerapan logika if-else menggunakan Lambda Expression. Jika mapper findById(id) tidak mengembalikan sesuatu, jalankan method orElseThrow() lalu kembalikan sebuah exception ResourceNotFoundException.*

1. **New Date/Time API**

* **LocalDateTime**

Menampilkan data tanggal dan waktu tanpa timezone.

**\*contoh:**

|  |
| --- |
| LocalDateTime currentDateTime = LocalDateTime.now();  System.out.println(“Current DateTime: ” + currentDateTime); |

**\*output:** Current DateTime: 2020-04-06T09:44:21.763

* **ZonedDateTime**

Menampilkan data tanggal dan waktu dengan timezone.

**\*contoh:**

|  |
| --- |
| ZonedDateTime currentDt = ZonedDateTime.now(ZoneId.of("Europe/Paris")); System.out.println("Current DateTime w/ Timezone: " + zonedDateTimeTester()); |

**\*output:** Current DateTime w/ Timezone: 2020-04-06T04:53:29.680+02:00[Europe/Paris]

* **Period**

Menampilkan jumlah tanggal diantara 2 (dua) tanggal

**\*contoh:**

|  |
| --- |
| LocalDate currentDate = LocalDate.now();  LocalDate independenceDate = LocalDate.of(2020, Month.AUGUST, 17);  Period datePeriod = Period.between(currentDate, independenceDate);  System.out.println(periodTester().toTotalMonths() + " months later to Indonesia Independence Day"); |

**\*output:** 4 months to Indonesia Independence Day

* **Duration**

Menampilkan total waktu dalam jam (hour), menit (minute), atau detik (second) diantara 2 (dua) waktu.

**\*contoh:**

|  |
| --- |
| LocalTime currentTime = LocalTime.now();  LocalTime myDateTime = LocalTime.of(18, 30);  Duration timeDuration = Duration.between(currentTime, myDateTime);  System.out.println(durationTester().toHours() + " hours later to Date w/ Your Wife"); |

**\*output:** 8 hours to Date w/ Your Wife

1. **Optional Class**

Fungsi class Optional yaitu untuk menunjukkan sebuah value ada atau tidak.

Keuntungan menggunakan Optional:

* Tidak perlu melakukan Null check
* Tidak usah khawatir dengan error NullPointerException
* Code menjadi lebih bersih

**\*contoh:**

|  |
| --- |
| Optional<String> name = Optional.of("Firman");  System.out.println("My name (Optional): " + optionalTester());  System.out.println("My name w/ value: " + optionalTester().get()); |

**\*output:**   
My name (Optional): Optional[Firman]

My name w/ value: Firman

**\*tips:** Jika sebuah properti memiliki kemungkinan untuk empty atau bernilai null, gunakan method ***Optional.ofNullable()*** untuk memberitahu class Optional jika sebuah property memungkinkan bernilai null. Class Optional akan menghandle value null tadi dengan menggantinya menjadi objek Optional.empty(). Jika kamu tetap bersikeras menggunakan method ***Optional.of()***, yang akan kamu dapatkan adalah sebuah java.lang.NullPointerException, dan ya aplikasi kamu akan berhenti, ingat ya!

1. **Streams**

Fungsi class ini yaitu memproses data sequence (antrian) dari sebuah objek. Data antrian ini nantinya dapat di filter dan di map. Objek Stream dapat dibuat dari array objek atau collection dengan bantuan method stream() dan atau of().

**\*contoh dengan stream():**

|  |
| --- |
| String[] strArr = new String[] {"a", "b", "c", "d"};  Stream<String> stream = Arrays.stream(strArr);  System.out.println(stream.collect(Collectors.toList())); |

**\*output:** [a, b, c, d]

**\*contoh dengan of():**

|  |
| --- |
| Stream<String> stream = Stream.empty();  stream.of(“a”, “b”, “c”);  System.out.println(stream.collect(Collectors.toList())); |

**\*output:** [a, b, c]

**\*contoh inisiasi Stream objek dari ArrayList:**

|  |
| --- |
| List<String> jobs = new ArrayList<>();  jobs.add("IT Consultant");  jobs.add("Software Developer");  jobs.add("Backend Developer");  jobs.add("Frontend Developer");  jobs.add("IT Support");    Stream<String> stream = jobs.stream();  System.out.println(stream.collect(Collections.toList())); |

**\*output:** [IT Consultant, Software Developer, Backend Developer, Frontend Developer, IT Support]

**\*contoh memfilter Stream:**

|  |
| --- |
| List<String> jobs = new ArrayList<>();  jobs.add("IT Consultant");  jobs.add("Software Developer");  jobs.add("Backend Developer");  jobs.add("Frontend Developer");  jobs.add("IT Support");    Stream<String> stream = jobs.stream().filter(job -> job.contains("Dev"));  System.out.println(stream.collect(Collections.toList())); |

**\*output:** [Software Developer, Backend Developer, Frontend Developer]

***-end-***