
Membuat Project



Membuat Project

- <https://start.spring.io/>
- Minimal Java 21

—

Record



Java Record

- Java Record adalah fitur baru yang dirilis pada versi Java 14, dan stabil di versi Java 16
- <https://openjdk.org/jeps/395>



Sebelum Java Records

- Sebelum adanya Java Record, setiap representasi data, kita akan buat dalam bentuk Java Bean Class
- Dimana terdapat Class dengan Getter Setter, Equals Method dan hashCode Method
- Pembuatan Java Bean Class untuk kasus-kasus sederhana terlalu bertele-tele dan terlalu banyak ceremony nya



Kode : Point Class

```
2 usages  new *
public class Point {

    6 usages
    private int x;

    6 usages
    private int y;

    no usages  new *
    @Contract(pure = true)
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
no usages  new *
public int getX() {
    return x;
}

no usages  new *
public void setX(int x) {
    this.x = x;
}

no usages  new *
public int getY() {
    return y;
}

no usages  new *
public void setY(int y) {
    this.y = y;
}
}
```

```
new *
@Contract(value = "null -> false", pure = true)
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Point point = (Point) o;

    if (x != point.x) return false;
    return y == point.y;
}

new *
@Override
public int hashCode() {
    int result = x;
    result = 31 * result + y;
    return result;
}
}
```



Immutable Data

- Saat membuat Data, kita sering membuat Immutable Data, yaitu Data yang tidak akan pernah berubah lagi
- Biasanya, pada kasus ini, kita akan membuat Class dengan Constructor, lalu menghapus semua method Setter nya, dan membuat semua property-nya menjadi final

Kode : Point Class

```
public class Point {  
  
    5 usages  
    private final int x;  
  
    5 usages  
    private final int y;  
  
    no usages    new *  
    @Contract(pure = true)  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

```
no usages    new *  
public int getX() {  
    return x;  
}
```

```
no usages    new *  
public int getY() {  
    return y;  
}
```

```
@Contract(value = "null -> false", pure = true)  
@Override  
public boolean equals(Object o) {  
    if (this == o) return true;  
    if (o == null || getClass() != o.getClass()) return false;  
  
    Point point = (Point) o;  
  
    if (x != point.x) return false;  
    return y == point.y;  
}  
  
new *  
@Override  
public int hashCode() {  
    int result = x;  
    result = 31 * result + y;  
    return result;  
}
```




Record

- Record adalah fitur di Java yang digunakan untuk membuat Immutable Data
- Record mirip seperti Class, hanya saja sifatnya adalah immutable, tidak bisa berubah lagi isi datanya setelah dibuat
- Artinya, property di Record adalah final (tidak bisa berubah), dan tidak ada method untuk mengubah data di record
- Record juga secara otomatis akan membuatkan Equals Method, hashCode Method dan ToString Method dari semua property yang terdapat di Record



Kode : Record Point

```
no usages new *  
public record Point(int x, int y) {  
}
```

—

Property



Record Property

- Saat membuat Record, kita perlu sebutkan langsung Property yang ingin kita tambahkan pada bagian kurung buka dan kurung tutup pada Record
- Hal ini secara otomatis menjadi Constructor, sehingga ketika kita membuat Objectnya, kita harus mengisi data property tersebut



Kode : Record Customer

```
1 usage    new *  
public record Customer(String id, String name, String email, String phone) {  
}💡
```



Kode : Membuat Customer

```
new
@Test
void createNewRecord() {
    var customer = new Customer("1", "Eko", "eko@localhost", "088888");
    assertNotNull(customer);

    System.out.println(customer);
}
```



Mengakses Property

- Property di record, selain final dia juga private
- Lantas bagaimana jika kita ingin mengakses data property dari instance yang sudah dibuat?
- Record secara otomatis akan membuat method dengan nama yang sama dengan property, misal untuk id bernama id(), untuk name bernama name(), untuk firstName bernama firstName(), dan seterusnya
- Method di record tidak dibuat dalam bentuk Getter



Kode : Get Record Property

```
@Test
void getRecordProperty() {
    var customer = new Customer("1", "Eko", "eko@localhost", "088888");
    assertEquals("1", customer.id());
    assertEquals("Eko", customer.name());
    assertEquals("eko@localhost", customer.email());
    assertEquals("088888", customer.phone());
}
```

Constructor



Record Constructor

- Sama seperti class biasanya, record juga bisa memiliki constructor lainnya
- Namun, setiap kita membuat constructor, kita wajib memanggil constructor utama yang dibuat pada record



Kode : Record Constructor

```
public record Customer(String id, String name, String email, String phone) {
```

```
    1 usage    new *
```

```
    public Customer(String id, String name, String email) {  
        this(id, name, email, null);  
    }
```

```
    no usages    new *
```

```
    public Customer(String id, String name) {  
        this(id, name, null);  
    }
```



Kode : Test Record Constructor

```
@Test
void recordConstructor() {
    var customer = new Customer("1", "Eko", "eko@localhost");
    assertEquals("1", customer.id());
    assertEquals("Eko", customer.name());
    assertEquals("eko@localhost", customer.email());
    assertNull(customer.phone());
}
```



Canonical Constructor

- Record bisa memiliki constructor yang parameter nya sama dalam definisi Record, atau disebut canonical constructor
- Secara otomatis canonical constructor ini akan dipanggil ketika Object dibuat
- Di dalam canonical constructor, kita wajib menginisialisasi seluruh property, karena seluruh property bersifat final



Kode : Canonical Constructor

```
4 usages    new *
public record Customer(String id, String name, String email, String phone) {

    3 usages    new *
    public Customer( @NotNull String id, String name, String email, String phone) {
        System.out.println("Create Customer");
        this.id = id.toLowerCase();
        this.name = name;
        this.email = email != null ? email.toLowerCase() : null;
        this.phone = phone != null ? phone.toLowerCase() : null;
    }
}
```



Kode : Test Canonical Constructor

```
@Test
void canonicalConstructor() {
    var customer = new Customer("1", "Eko", "EKO@LOCALHOST");
    assertEquals("1", customer.id());
    assertEquals("Eko", customer.name());
    assertEquals("eko@localhost", customer.email());
}
```




Compact Constructor

- Kadang, saat membuat canonical constructor, kita tidak ingin mengubah property apapun, yang artinya data property hanya langsung diberikan dari parameter pada constructor
- Pada kasus seperti ini, dibanding menggunakan canonical constructor, kita bisa menggunakan compact constructor
- Compact constructor adalah constructor yang tidak perlu mendeklarasikan parameter, secara otomatis parameter akan digunakan sebagai nilai untuk property



Kode : Compact Constructor

```
public record Point(int x, int y) {  
    no usages    new *  
    public Point {  
        System.out.println("Create Point");  
    }  
}
```





Kode : Compact Constructor Test

```
@Test
void compactConstructor() {
    var point = new Point(10, 10);
    assertEquals(10, point.x());
    assertEquals(10, point.y());
}
```

Method



Record Method

- Record juga memiliki kemampuan untuk membuat method seperti class biasanya
- Oleh karena itu, kita bisa menambah method sama seperti di class biasanya



Kode : Record Method

```
public record Customer(String id, String name, String email, String phone) {  
  
    public String sayHello(String name) {  
        return "Hello " + name + ", my name is " + this.name();  
    }  
}
```



Kode : Test Record Method

```
@Test
void recordMethod() {
    var customer = new Customer("1", "Eko", "eko@localhost");
    assertEquals("Hello Budi, my name is Eko", customer.sayHello("Budi"));
}
```

Equals, Hash Code dan ToString



Equals, Hash Code dan ToString

- Salah satu fitur yang terdapat di record adalah, kemampuan membuat method `equals()`, `hashCode()` dan `toString()` secara otomatis
- Secara otomatis record akan membuat method `equals()`, `hashCode()` dan `toString()` yang menggunakan seluruh property yang terdapat di record
- Namun, jika kita ingin melakukan override, kita juga bisa membuat method tersebut secara manual



Kode : Record Test

```
@Test
void recordEquals() {
    var customer1 = new Customer("1", "Eko", "eko@localhost");
    var customer2 = new Customer("1", "Eko", "eko@localhost");

    assertTrue(customer1.equals(customer2));
    assertEquals(customer1.hashCode(), customer2.hashCode());
    assertEquals(customer1.toString(), customer2.toString());
}
```

Inheritance



Record / Class Inheritance

- Record tidak bisa melakukan extends ke class atau record lainnya
- Record mirip seperti Enum, dimana secara otomatis akan di set sebagai class turunan class tertentu
- Enum secara otomatis akan menjadi class turunan dari `java.lang.Enum`
- Record secara otomatis akan menjadi class turunan dari `java.lang.Record`
- Selain itu, Record merupakan class yang final, artinya tidak bisa diturunkan lagi oleh class lainnya
- Oleh karena itu, Record memang cocok sebagai representasi dari Immutable Data



Interface Inheritance

- Namun, record bisa melakukan pewarisan dari Interface
- Sama seperti pada class biasanya



Kode : Interface SayHello

```
public interface SayHello {  
  
    String sayHello(String name);  
  
}
```





Kode : Interface Inheritance

```
public record Customer(String id, String name, String email, String phone)
    implements SayHello {

    public String sayHello(String name) {
        return "Hello " + name + ", my name is " + this.name();
    }
}
```

Static



Static

- Record sama seperti class biasanya, bisa memiliki static field dan method
- Hal ini dikarenakan static bukanlah anggota dari instance, oleh karena itu, static masih bisa dibuat di record



Kode : Static

```
public record Point(int x, int y) {  
    public Point {  
        System.out.println("Create Point");  
    }  
  
    public static Point ZERO = new Point(0, 0);  
  
    public static Point create(int x, int y) {  
        return new Point(x, y);  
    }  
}
```



Kode : Test Static

```
@Test
void staticMember() {
    assertEquals(0, Point.ZERO.x());
    assertEquals(0, Point.ZERO.y());

    var point = Point.create(10, 10);
    assertEquals(10, point.x());
    assertEquals(10, point.y());
}
```

—

Generic



Generic

- Record juga bisa menggunakan fitur Java Generic, sama seperti class biasanya



Kode : Generic

```
public record Data<T>(T data ) {
```

```
}  
=
```





Kode : Test Generic

```
@Test
void generic() {
    var data = new Data<String>("Eko");
    assertEquals("Eko", data.data());

    var data2 = new Data<Integer>(100);
    assertEquals(100, data2.data());
}
```

Annotation



Annotation

- Record juga bisa menggunakan annotation sama seperti class biasanya
- Yang membedakan adalah ketika kita menambahkan annotation ke property pada record
- Secara otomatis annotation tersebut akan ditambahkan pada constructor parameter, property/field dan juga method (getter)



Kode : Valid Annotation

```
✓ import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

✓ @Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER})
  @Retention(RetentionPolicy.RUNTIME)
  @Documented
  public @interface Valid {
  }💡
```



Kode : Record Annotation

```
public record Point(@Valid int x, @Valid int y) {  
    public Point {  
        System.out.println("Create Point");  
    }  
  
    public static Point ZERO = new Point(0, 0);  
  
    public static Point create(int x, int y) {  
        return new Point(x, y);  
    }  
}
```



Kode : Test Annotation

```
@Test
void annotation() throws NoSuchMethodException, NoSuchFieldException {
    var point = new Point(10, 10);
    assertNotNull(point.getClass().getDeclaredField("x").getAnnotation(Valid.class));
    assertNotNull(point.getClass().getDeclaredField("y").getAnnotation(Valid.class));

    assertNotNull(point.getClass().getDeclaredMethod("x").getAnnotation(Valid.class));
    assertNotNull(point.getClass().getDeclaredMethod("y").getAnnotation(Valid.class));

    assertNotNull(point.getClass().getConstructors()[0].getParameters()[0].getAnnotation(Valid.class));
    assertNotNull(point.getClass().getConstructors()[0].getParameters()[1].getAnnotation(Valid.class));
}
```

Reflection



Reflection

- Dengan adanya fitur record, `java.lang.Class` memiliki method baru bernama `getRecordComponents()`, ini digunakan untuk mendapatkan data `java.lang.reflect.RecordComponent`
- `RecordComponent` berisi informasi property dari record
- Untuk mengecek apakah class ini record, kita bisa menggunakan method `isRecord()` pada `java.lang.Class`



Kode : Reflection

```
@Test
void recordComponent() {
    var customer = new Customer("1", "Eko", "eko@localhost");

    assertTrue(customer.getClass().isRecord());

    RecordComponent[] components = customer.getClass().getRecordComponents();
    assertEquals(4, components.length);
}
```

Record Patterns



Record Patterns

- Java 21 memperkenalkan fitur bernama Record Patterns, sebuah fitur untuk mempermudah kita ketika membongkar isi dari Record
- Fitur ini sebenarnya sudah diperkenalkan sejak Java 19, namun baru rilis di versi Java 21
- <https://openjdk.org/jeps/440>



Sebelum Record Patterns

- Sebelum adanya fitur Record Patterns, setiap kali kita ingin melakukan pengecekan apakah sebuah object adalah instance of Record, maka kita harus konversi secara manual
- Lalu mengambil data component di record dari object hasil konversi



Kode : Sebelum Record Patterns

```
public void printObject(Object object) {  
    if (object instanceof Point) {  
        Point point = (Point) object;  
        System.out.println(point.x());  
        System.out.println(point.y());  
    } else {  
        System.out.println(object);  
    }  
}  
  
@Test  
void beforeRecordPatterns() {  
    printObject(new Point(10, 10));  
}
```



Dengan Record Patterns

- Dengan Record Patterns, kita bisa secara otomatis langsung melakukan extract data component ke variabel saat menggunakan instanceof



Code : Record Patterns

```
public void printObject(Object object) {  
    if (object instanceof Point(int x, int y)) {  
        System.out.println(x);  
        System.out.println(y);  
    } else {  
        System.out.println(object);  
    }  
}  
  
@Test  
void beforeRecordPatterns() {  
    var point = new Point(10, 10);  
    printObject(point);  
}
```



Nested Record Patterns

- Record Patterns juga bisa digunakan untuk tipe Record yang nested



Kode : Line Record

```
public record Line(Point start, Point end) {
```

```
}   
|
```



Kode : Line Record Patterns

```
public void printObject(Object object) {  
    if (object instanceof Line(Point(int x, int y), Point end)) {  
        System.out.println(x);  
        System.out.println(y);  
        System.out.println(end);  
    } else if (object instanceof Point(int x, int y)) {  
        System.out.println(x);  
        System.out.println(y);  
    } else {  
        System.out.println(object);  
    }  
}
```