

---

# Pengenalan Java Reflection

---

# Pengenalan Java Reflection

- Java Reflection merupakan fitur Java dimana memperbolehkan program Java untuk mempelajari atau memodifikasi dirinya sendiri
- Misal saja kita bisa melihat struktur sebuah Java class dari mulai semua fields, method, constructor dan lain-lain saat aplikasi berjalan
- Bahkan Java Reflection bisa digunakan untuk memodifikasi kode program yang sedang berjalan
- Java Reflection banyak sekali digunakan oleh framework-framework karena memang sangat powerfull sekali

---

# Package Java Reflection

- Fitur Java Reflection terdapat dalam dua package, yaitu java.lang dan java.lang.reflect
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/package-summary.html>
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/package-summary.html>
- Ada banyak sekali class yang terdapat dalam Java Reflection, dan akan kita coba bahas satu persatu dalam kelas ini

---

# Membuat Project

---

# Membuat Project

<https://start.spring.io/>

---

# Class

---

## Class<T>

- `java.lang.Class<T>` merupakan representasi dari reflection untuk Java Class, Interface dan Enum
- Saat kita membuat Java Class, Interface atau Enum, kadang kita menambahkan field dan method
- Dengan kemampuan `java.lang.Class`, kita bisa membaca seluruh data member yang terdapat pada Java Class, Interface atau Enum pada saat aplikasi nya berjalan
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Class.html>

---

## Membuat Class<T>

- Untuk membuat Class<T>, kita bisa melakukan beberapa cara
- Cara yang pertama adalah dengan menggunakan kata kunci .class setelah nama Java Class, Interface atau Enum nya, misal Person.class, Repository.class, atau Gender.class
- Atau kita juga bisa membuat Class<T> dari sebuah String, menggunakan static method Class.forName("com.example.blabla.Person")
- Atau kita juga bisa mengambil Class<T> dari object, dengan menggunakan method getClass()

---

# Kode : Class Person

```
public class Person {  
  
    private String firstName;  
  
    private String lastName;  
  
    public Person() {  
    }  
  
    public Person(String firstName, String lastName) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
}
```



## Kode : Membuat Class<T>

```
@Test  
void testClass() throws ClassNotFoundException {  
  
    Class<Person> personClass1 = Person.class;  
  
    Class<?> personClass2 = Class.forName("programmerzamannow.reflection.Person");  
}
```

---

# Class<T> Method

- Class<T> memiliki banyak sekali method
- Seperti untuk mendapatkan method, fields, constructor, annotation, superclass, interfaces dan lain-lain
- Semua detail method nya bisa kita baca di Javadoc
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Class.html>



# Kode : Class<T> Method

```
@Test
void classMethod() {
    Class<Person> personClass = Person.class;

    System.out.println(personClass.getSuperclass());
    System.out.println(personClass.getInterfaces());
    System.out.println(personClass.getDeclaredConstructors());
    System.out.println(personClass.getDeclaredMethods());
    System.out.println(personClass.getDeclaredMethods());
    System.out.println(personClass.getModifiers());
    System.out.println(personClass.getPackage());
    System.out.println(personClass.getName());
}
```

---

# Field



# Field

- Field merupakan representasi dari Java Field yang terdapat di dalam Java Class
- Untuk mendapatkan public Field, kita bisa menggunakan method getFields()
- Atau jika ingin mendapatkan semua field dengan semua visibility, kita bisa menggunakan method getDeclaredFields()
- Atau kita juga bisa mendapatkan Field berdasarkan nama field nya menggunakan method getField(name) atau getDeclaredField(name)
- Field sama seperti Class<T>, memiliki banyak sekali method yang bisa kita gunakan untuk melihat detail dari Field tersebut, seperti tipe data, nama field, annotation, dan lain-lain
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Field.html>



## Kode : Field

```
Class<Person> personClass = Person.class;

Field[] fields = personClass.getDeclaredFields();
for (Field field : fields) {
    System.out.println(field.getName() + " : " + field.getType().getName());
}
```

---

# Mengambil atau Mengubah Field Object

- Field memiliki kemampuan untuk mengambil atau mengubah field dari object yang ada
- Misal kita sudah membuat Field, lalu kita memiliki object person1, lalu kita ingin mengambil nilai Field tersebut atau mengubahnya, kita bisa menggunakan method setXxx() atau getXxx() pada Field



## Kode : Mengambil Field Object

```
Person person = new Person("Eko", "Khannedy");

Class<Person> personClass = Person.class;
Field firstName = personClass.getDeclaredField(name: "firstName");
firstName.setAccessible(true);

Object firstNameValue = firstName.get(person);
System.out.println(firstNameValue);
```



## Kode : Mengubah Field Object

```
Person person = new Person("Eko", "Khannedy");

Class<Person> personClass = Person.class;
Field firstName = personClass.getDeclaredField( name: "firstName");
firstName.setAccessible(true);

firstName.set(person, "Joko");
System.out.println(person.getFirstName());
```

---

# Method



# Method

- Selain Field, kita juga bisa mendapatkan Method yang tersedia di Class<T>
- Cara mendapatkannya pun sama seperti Field, kita bisa menggunakan method getMethods(), getDeclaredMethods(), getMethod(name) dan getDeclaredMethod(name)
- Method pun banyak sekali method yang bisa kita gunakan untuk mendapatkan informasi seperti return value, name, annotation, parameter dan lain-lain
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Method.html>



# Kode : Method

```
Class<Person> personClass = Person.class;

Method[] methods = personClass.getDeclaredMethods();
for (Method method : methods) {
    System.out.println(method.getName());
    System.out.println(method.getReturnType());
    System.out.println(toString(method.getParameterTypes()));
    System.out.println(toString(method.getExceptionTypes()));
    System.out.println("=====");
}
```

---

# Memanggil Method Object

- Method bisa digunakan untuk memanggil method pada sebuah object
- Hampir mirip dengan Field yang bisa digunakan untuk mengambil atau mengubah field didalam object
- Untuk memanggil method object, kita bisa menggunakan method invoke(object, parameters...)



## Kode : Memanggil Method Object

```
Person person = new Person("Eko", "Kurniawan");

Class<Person> personClass = Person.class;
Method getFirstName = personClass.getDeclaredMethod( name: "getFirstName");

Object response = getFirstName.invoke(person);
System.out.println(response);
```

---

# Parameter



# Parameter

- Parameter merupakan representasi dari Java Parameter di Java Method
- Cara mendapatkan Parameter, kita bisa ambil dari Method, karena Parameter memang hanya ada di Method dan Constructor (yang akan kita bahas nanti)
- Parameter memiliki banyak sekali method, seperti untuk mendapatkan tipe data parameter, nama parameter, dan lain-lain
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Parameter.html>



# Kode : Parameter

```
Class<Person> personClass = Person.class;

for (Method method : personClass.getDeclaredMethods()) {
    Parameter[] parameters = method.getParameters();
    System.out.println(method.getName());
    for (Parameter parameter : parameters) {
        System.out.println("Parameter Name : " + parameter.getName());
        System.out.println("Parameter Type : " + parameter.getType());
    }
    System.out.println("=====");
}
```

---

# Memanggil Method Object dengan Parameter

- Sama seperti Method tanpa parameter
- Kita juga bisa memanggil Method yang memiliki parameter



# Kode : Memanggil Method dengan Parameter

```
Class<Person> personClass = Person.class;
Method setFirstName = personClass.getDeclaredMethod( name: "setFirstName", String.class);

Person person = new Person("Eko", "Khannedy");
setFirstName.invoke(person, ...args: "Joko");

System.out.println(person.getFirstName());
```

---

# Constructor

---

# Constructor<T>

- Constructor<T> merupakan representasi dari Java Constructor di Java Class
- Constructor<T> ini mirip dengan Method, dimana dia memiliki Parameter
- Untuk membuat Constructor kita mendapatkannya melalui Class<T>
- Constructor<T> merupakan tipe data generic, mengikuti tipe data generic dari Class<T> nya
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Constructor.html>



# Kode : Constructor

```
Class<Person> personClass = Person.class;

for (Constructor<?> constructor : personClass.getDeclaredConstructors()) {
    System.out.println(constructor.getName());
    for (Parameter parameter : constructor.getParameters()) {
        System.out.println(parameter.getName());
        System.out.println(parameter.getType());
    }
}
```

---

# Membuat Object dengan Constructor

- Kita sudah tahu bahwa Constructor merupakan method yang dieksekusi ketika sebuah Object pertama kali dibuat
- Dengan menggunakan Constructor, kita juga bisa membuat object baru
- Caranya dengan menggunakan method newInstance(parameter...)



# Kode : Membuat Object dengan Constructor

```
Class<Person> personClass = Person.class;
Constructor<Person> constructorEmpty = personClass.getConstructor();
Constructor<Person> constructorParameters = personClass.getConstructor(String.class, String.class);

Person person1 = constructorEmpty.newInstance();
Person person2 = constructorParameters.newInstance(...initargs: "Eko", "Khannedy");

System.out.println(person1);
System.out.println(person2);
```

---

# Super Class

---

# Super Class

- Dengan menggunakan Java Reflection, kita juga bisa mengetahui Super Class dari sebuah Java Class
- Terdapat method `getSuperclass()` di `Class<T>` untuk mendapatkan Super Class nya
- Perlu diingat, bahwa saat kita membuat class, jika kita tidak menambahkan super class, secara otomatis super class nya adalah class `java.lang.Object`
- [https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Class.html#getSuperclass\(\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Class.html#getSuperclass())



## Kode : Super Class

```
Class<Person> personClass = Person.class;  
System.out.println(personClass); // Person
```

```
Class<? super Person> superclass = personClass.getSuperclass();  
System.out.println(superclass); // Object
```

```
Class<? super Person> superclassTop = superclass.getSuperclass();  
System.out.println(superclassTop); // null
```

---

# Interface



# Interface

- Sebelumnya kita hanya bahas tentang Class, bagaimana dengan Interface?
- Interface sebenarnya representasi di Java Reflection tetaplah Class<T>
- Yang membedakan adalah, pada Interface sudah pasti tidak memiliki Constructor dan juga tidak bisa
- Selain itu untuk mengetahui apakah Class<T> ini adalah Java Class atau Java Interface, kita bisa menggunakan method isInterface()



## Kode : Nameable Interface

```
public interface Nameable {  
  
    String getFirstName();  
  
    String getLastName();  
  
}
```



## Kode : Interface

```
Class<Nameable> nameableClass = Nameable.class;  
  
System.out.println(nameableClass.isInterface());  
System.out.println(nameableClass.getSuperclass());
```

---

# Super Interface

- Kita tahu bahwa Interface juga mendukung pewarisan, berbeda dengan class, saat kita melakukan implements Interface, bisa lebih dari satu interface, selain itu interface juga bisa extends Interface
- Pada kasus seperti ini, jika kita melihat semua super interface Class<T>, kita bisa menggunakan method getInterfaces()

---

## Kode : Implement Interface

```
public class Person implements Nameable {  
    private String firstName;  
    private String lastName;  
    public Person() {  
    }  
}
```



## Kode : Super Interface

```
Class<Person> personClass = Person.class;

for (Class<?> anInterface : personClass.getInterfaces()) {
    System.out.println(anInterface.getName());
}
```

---

# Modifier



# Modifier

- Modifier merupakan representasi dari Java Modifier, seperti misal private, protected, public, abstract, dan lain-lain
- Modifier di Java Reflection direpresentasikan oleh number int, dan untuk mempermudah, kita bisa menggunakan class Modifier untuk mengecek jenis modifier nya
- Untuk mendapatkan data modifier, kita bisa menggunakan getModifiers(), entah itu di class, method, field, parameter, constructor, dan lain-lain
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Modifier.html>



## Kode : Modifier

```
Class<Person> personClass = Person.class;  
  
System.out.println(Modifier.isPublic(personClass.getModifiers()));  
System.out.println(Modifier.isFinal(personClass.getModifiers()));  
System.out.println(Modifier.isStatic(personClass.getModifiers()));
```

---

# Package



# Package

- Package merupakan representasi dari Java Package
- Kita bisa mendapatkan Package dari Class<T> dengan menggunakan method getPackage()
- Terdapat banyak informasi di dalam Package yang bisa kita gunakan
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Package.html>



## Kode : Package

```
Class<Person> personClass = Person.class;  
  
Package aPackage = personClass.getPackage();  
System.out.println(aPackage.getName());  
System.out.println(Arrays.toString(aPackage.getAnnotations()));
```

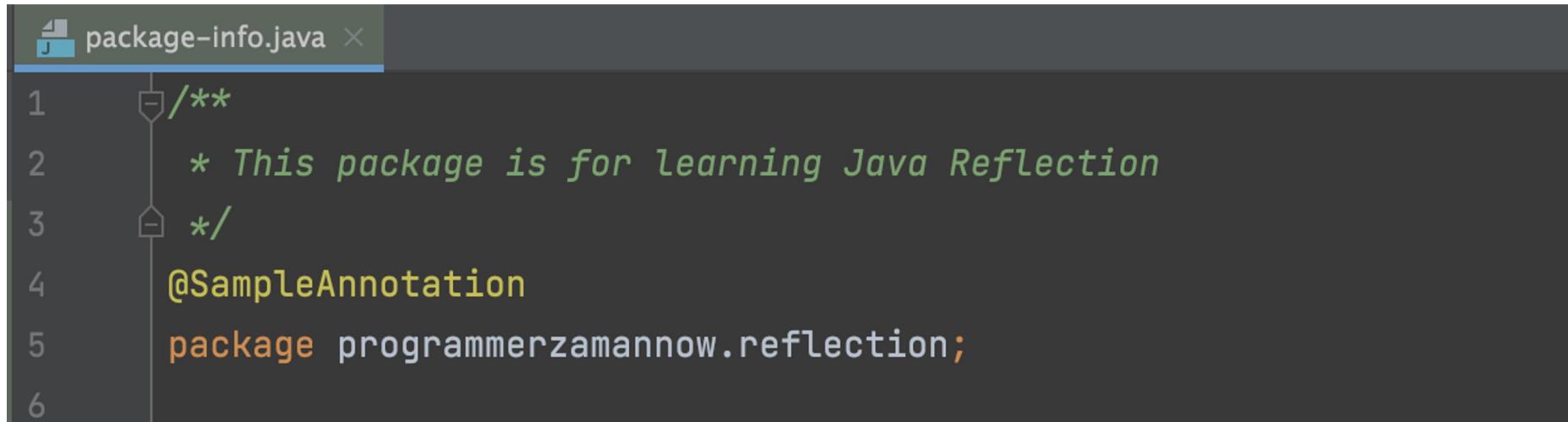
---

# **package-info.java**

- Package di Java bisa kita tambahkan informasi tambahan
- Seperti javadoc dan annotation misalnya
- Caranya kita bisa membuat file package-info.java di package yang kita inginkan
- Lalu kita tambahkan informasi yang kita mau pada package tersebut

---

## Kode : package-info.java



A screenshot of a Java code editor showing the file `package-info.java`. The code contains a package declaration with annotations and a multi-line comment.

```
1  */
2  * This package is for learning Java Reflection
3  */
4  @SampleAnnotation
5  package programmerzamannow.reflection;
6
```

---

# Annotation

---

# Annotation

- Pada Kelas Java OOP kita sudah bahas tentang Annotation
- Sekarang kita bahas tentang mendapatkan informasi Annotation dengan menggunakan Reflection
- Annotation merupakan fitur yang sangat powerfull sekali di Java, banyak sekali framework menggunakan Annotation
- Annotation bisa ditempatkan dimanapun, di class, method, field, constructor, parameter, package dan lain-lain
- Setiap kita membuat Annotation, secara tidak langsung kita telah membuat turunan `java.lang.annotation.Annotation`
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Annotation.html>

---

## Kode : @NotBlank Annotation

```
import java.lang.annotation.*;

@Documented
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface NotBlank {

    boolean allowEmptyString() default false;

}
```

---

## Kode : Menggunakan Annotation

```
public class Person implements Nameable {  
  
    @NotBlank  
    private String firstName;  
  
    @NotBlank(allowEmptyString = true)  
    private String lastName;
```

# Kode : Validate Not Blank

```
public class Validator {  
  
    public static void validateNotBlank(Object person) throws IllegalAccessException {  
        Class<?> aClass = person.getClass();  
        for (Field field : aClass.getDeclaredFields()) {  
            field.setAccessible(true);  
            NotBlank notBlank = field.getAnnotation(NotBlank.class);  
            if (notBlank != null) {  
                String value = (String) field.get(person);  
                if (!notBlank.allowEmptyString()) value = value.trim();  
                if (value == null || value.equals("")) {  
                    throw new RuntimeException("Field : " + field.getName() + " is blank");  
                }  
            }  
        }  
    }  
}
```

---

## Kode : Test Annotation

```
Person person = new Person("Eko", "");  
Validator.validateNotBlank(person);
```

```
}
```

---

# Enum

---

# Enum

- Enum di Java Reflection, sama seperti Interface, direpresentasikan dengan Class<T>, hal ini dikarenakan, pada Enum juga kita bisa menambahkan field, method dan constructor
- Yang membedakan adalah, method `isEnum()` nya bernilai true
- Dan untuk mendapatkan semua nilai Enum, kita bisa menggunakan method `getEnumConstants()`
- Selain itu super class Enum adalah `java.lang.Enum`
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Enum.html>



# Kode : Gender Enum

```
public enum Gender {  
  
    MALE(description: "Male"),  
    FEMALE(description: "Female");  
  
    private final String description;  
  
    Gender(String description) {  
        this.description = description;  
    }  
}
```



## Kode : Enum

```
Class<Gender> genderClass = Gender.class;  
  
System.out.println(genderClass.isEnum());  
System.out.println(genderClass.getSuperclass());  
System.out.println(Arrays.toString(genderClass.getEnumConstants()));
```

---

# Primitive Type

---

# Primitive Type

- Apa yang terjadi jika kita memiliki field atau parameter atau method yang mengembalikan nilai primitive type? Seperti int, long, boolean, dan lain-lain
- Data tersebut pun, pada Java Reflection tetap direpresentasikan dalam Class<T>
- Untuk membuat Class<T> primitive, kita langsung gunakan .class setelah tipe data primitive tersebut, namun Java akan secara otomatis mengkonversi nya menjadi tipe data non primitive, misal int menjadi Integer, boolean menjadi Boolean, dan lain-lain
- Namun yang membedakan adalah, method isPrimitive() akan bernilai true untuk tipe data primitive



## Kode : Primitive Type

```
Class<Integer> integerClass = int.class;
Class<Long> longClass = long.class;
Class<Boolean> booleanClass = boolean.class;

System.out.println(integerClass.isPrimitive());
System.out.println(longClass.isPrimitive());
System.out.println(booleanClass.isPrimitive());
```

---

# Mengambil Data Primitive Type

- Khusus tipe data primitive, ketika ingin mengambil data di Field, kita bisa gunakan method getXxx() sesuai dengan tipe data nya, misal getInt(), getBoolean() dan lain-lain
- Namun ketika mengambil method dengan parameter, kita bisa gunakan representasi tipe data object nya, misal untuk int gunakan Integer, long gunakan Long, dan lain-lain



## Kode : Age di Person

```
public class Person implements Nameable {  
  
    @NotBlank  
    private String firstName;  
  
    @NotBlank(allowEmptyString = true)  
    private String lastName;  
  
    private int age;
```



## Kode : Field Primitive

```
Class<Person> personClass = Person.class;
Field age = personClass.getDeclaredField( name: "age");

System.out.println(age.getType().isPrimitive());
```



## Kode : Mengambil Field Data Primitive

```
Class<Person> personClass = Person.class;
Field age = personClass.getDeclaredField( name: "age");

Person person = new Person();
person.setAge(30);

age.setAccessible(true);
System.out.println(age.getInt(person));
```



## Kode : Mengambil Method Primitive Type

```
Class<Person> personClass = Person.class;
Method setAge = personClass.getDeclaredMethod( name: "setAge", int.class);

Person person = new Person();
setAge.invoke(person, ...args: 30);

System.out.println(person.getAge());
```

---

# Array

---

# Array

- Sama seperti tipe data yang lainnya, Array pun di representasikan dalam bentuk Class<T> di Java Reflection
- Untuk membuat Class<T> Array, kita bisa gunakan .class setelah Array nya, misal String[].class, int[].class, dan sebagainya
- Bahkan kita bisa buat array multi dimensi, misal String[][] .class
- Yang membedakan dengan Class<T> lainnya, pada Array, method isArray() nya akan bernilai true



## Kode : Array

```
Class<String[]> stringArrayClass = String[].class;
Class<int[][]> intArrayClass = int[][][].class;

System.out.println(stringArrayClass.isArray());
System.out.println(intArrayClass.isArray());
```

---

# Array Member

- Sedikit berbeda dengan tipe data seperti Class, Interface, Enum, pada Array tidak memiliki Class Member, seperti Field, Method dan Constructor
- Sehingga jika pada Class<T>, kita coba memanggil getFields(), getMethods(), getConstructors(), maka hasilnya adalah kosong



## Kode : Array Member

```
Class<String[]> stringArrayClass = String[].class;  
  
System.out.println(Arrays.toString(stringArrayClass.getDeclaredFields()));  
System.out.println(Arrays.toString(stringArrayClass.getDeclaredMethods()));  
System.out.println(Arrays.toString(stringArrayClass.getDeclaredConstructors()));
```

---

# **java.lang.reflect.Array**

- Lantas bagaimana jika kita ingin mengakses data array, membuat array dan lain-lain, jika pada Class<T>, semua class member nya tidak tersedia di Array?
- Untungnya terdapat class java.lang.reflect.Array , yang bisa kita gunakan untuk membantu menggunakan Class<T> dengan tipe Array
- Ada banyak method yang bisa kita gunakan, dari membuat array, mengakses datanya sampai mengubah data array
- Dan dari Class<T> Array, jika kita ingin tahu tipe data array nya, kita bisa gunakan method getComponentType()
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Array.html>



# Kode : java.lang.reflect.Array

```
Class<String[]> stringArrayClass = String[].class;

Object array = Array.newInstance(stringArrayClass.getComponentType(), length: 10);

Array.set(array, index: 0, value: "Eko");
Array.set(array, index: 1, value: "Kurniawan");

System.out.println(Array.get(array, index: 0));
System.out.println(Array.get(array, index: 1));
```

---

# Parameterized Type



# Parameterized Type

- Kita sudah hampir membahas semua jenis tipe Reflection di Java, namun di Java terdapat fitur yang bernama Generic Programming
- Bagaimana cara handle data generic di Java Reflection? misal List<String>, atau Map<String, String>
- Kita bisa menggunakan ParameterizedType untuk menangani hal ini
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/ParameterizedType.html>



## Kode : Menambah Generic Field

```
public class Person implements Nameable {  
  
    @NotBlank  
    private String firstName;  
  
    @NotBlank(allowEmptyString = true)  
    private String lastName;  
  
    private List<String> hobbies;
```



# Kode : Parameterized Return Type

```
Class<Person> personClass = Person.class;

Method getHobbies = personClass.getDeclaredMethod( name: "getHobbies");

Type returnType = getHobbies.getGenericReturnType();
if (returnType instanceof ParameterizedType) {
    ParameterizedType parameterizedType = (ParameterizedType) returnType;
    System.out.println(parameterizedType.getRawType());
    System.out.println(Arrays.toString(parameterizedType.getActualTypeArguments()));
}
```



# Kode : Parameterized Parameter Type

```
Class<Person> personClass = Person.class;

Method getHobbies = personClass.getDeclaredMethod( name: "setHobbies", List.class);
for (Type type : getHobbies.getGenericParameterTypes()) {
    if (type instanceof ParameterizedType) {
        ParameterizedType parameterizedType = (ParameterizedType) type;
        System.out.println(parameterizedType.getRawType());
        System.out.println(Arrays.toString(parameterizedType.getActualTypeArguments()));
    }
}
```



## Kode : Parameterized Field

```
Class<Person> personClass = Person.class;

Field hobbies = personClass.getDeclaredField( name: "hobbies");
Type type = hobbies.getGenericType();

if(type instanceof ParameterizedType) {
    ParameterizedType parameterizedType = (ParameterizedType) type;
    System.out.println(parameterizedType.getRawType());
    System.out.println(Arrays.toString(parameterizedType.getActualTypeArguments()));
}
```



# Type Variable

- Sebelumnya kita buat generic dengan data asli, seperti String dan lain-lain, bagaimana jika kita ingin mengetahui tipe data generic tanpa implementasi data asli, misal kita ingin mempelajari class List<T>, Map<K, V>
- Parameter generic tersebut, dalam Java Reflection, direpresentasikan dalam class TypeVariable
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/TypeVariable.html>



## Kode : Generic Class

```
public static class Data<T extends AutoCloseable> {  
  
    private T data;  
  
    public T getData() {  
        return data;  
    }  
}
```



## Kode : Type Variable di Class

```
Class<Data> dataClass = Data.class;

for (TypeVariable<Class<Data>> typeParameter : dataClass.getTypeParameters()) {
    System.out.println(typeParameter.getName());
    System.out.println(Arrays.toString(typeParameter.getBounds()));
}
```

---

# Proxy



# Proxy

- Saat kita belajar di kelas Java Unit Test, kita belajar dengan yang namanya stub, yaitu object tiruan
- Java Reflection memiliki fitur yang bernama Proxy
- Proxy bisa digunakan untuk membuat object bahkan tanpa harus membuat implementasi sebuah Interface
- Terlihat aneh, tapi fitur ini sangat berguna pada kasus-kasus tertentu, bahkan saat ini populer yang namanya Spring Data yang bisa digunakan untuk memanipulasi database cukup bermodalkan Interface, tanpa harus membuat implementasi class dan melakukan query database secara manual
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Proxy.html>

---

# Invocation Handler

- Saat kita membuat proxy, kita bisa secara dinamis menerima method yang dipanggil, dan mengembalikan value di method tersebut dengan InvocationHandler
- Fitur ini dalam bahasa pemrograman lain mirip sekali dengan yang namanya Meta Programming
- <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/InvocationHandler.html>

---

## Kode : Membuat Interface

```
public interface Car {  
    void run();  
    String getName();
```



## Kode : Membuat Invocation Handler

```
InvocationHandler invocationHandler = new InvocationHandler() {  
  
    @Override  
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
        if (method.getName().equals("getName")) {  
            return "Car Proxy";  
        } else if (method.getName().equals("run")) {  
            System.out.println("Car is running");  
        }  
        return null;  
    }  
}
```



## Kode : Membuat Proxy

```
Car car = (Car) Proxy.newProxyInstance(
    ClassLoader.getSystemClassLoader(),
    new Class[]{Car.class},
    invocationHandler
);

System.out.println(car.getName());

car.run();
```

---

# Record

---

# Record

- Di Java versi 16, fitur Java Record sudah stabil, dan kita juga bisa menggunakan Java Reflection
- Record di Java Reflection tetap direpresentasikan dengan Class<T>
- Hanya saja method isRecord() akan mengembalikan nilai true, selain itu untuk mendapatkan detail Record, kita bisa gunakan getRecordComponents()
- Dan jangan lupa, saat kita membuat Record, parent class nya bukanlah Object, melainkan java.lang.Record
- <https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/reflect/RecordComponent.html>

---

## Kode : Membuat Record

```
public record Product(String id, String name, Long price) {  
}
```



## Kode : Record

```
Class<Product> productClass = Product.class;

System.out.println(productClass.isRecord());
System.out.println(Arrays.toString(productClass.getDeclaredFields()));
System.out.println(Arrays.toString(productClass.getDeclaredMethods()));
System.out.println(Arrays.toString(productClass.getDeclaredConstructors()));
System.out.println(Arrays.toString(productClass.getRecordComponents()));
```



# Kode : Mengubah Record Component

```
Product product = new Product(id: "1", name: "iPhone", price: 20000000L);

Class<Product> productClass = Product.class;
RecordComponent component = Arrays.stream(productClass.getRecordComponents())
    .filter(recordComponent → recordComponent.getName().equals("id"))
    .findFirst().get();
Method method = component.getAccessor();

System.out.println(method.invoke(product));
```