**Java Reflection**

## **Pendahuluan**

**Sebelumnya harus sudah paham materi terkait** :

* Java Dasar
* Java Object Oriented Programming
* Apache Maven
* Java Unit Test

**Agenda**

* Pengenalan Java Reflection
* Class
* Fields
* Method
* Property
* Constructor
* Array
* Annotation
* Parameterized Type
* Dan lain-lain

## **Pengenalan Java Reflection**

* Java Reflection merupakan fitur Java dimana memperbolehkan program Java untuk mempelajari atau memodifikasi dirinya sendiri
* Misal saja kitab isa 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 baahs satu persatu dalam kelas ini

## **Class**

**Class<T>**

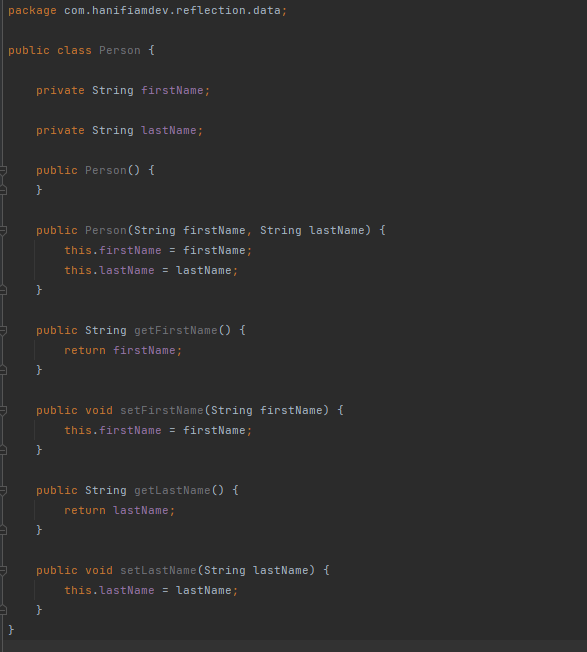
* Java.lang.Class<T> merupakan representasi dari reflection untuk Java Class, Interface dan Enum
* Saat kita membuat Java Class, Interface atau Enum, kadang kita menambahkan field dan method
* Dengan kemampuan java.lang.Class. kita bisa membaca seluruh data member yang terdapat pada Java Class, Interface 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>, kitab isa melakukan beberapa cara
* Cara yang pertama dengan meggunakan 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.forClass(“com.example.blabla.Person”)
* Atau kita juga bisa mengambil Class<T> dari object, dengan menggunakan method getClass()

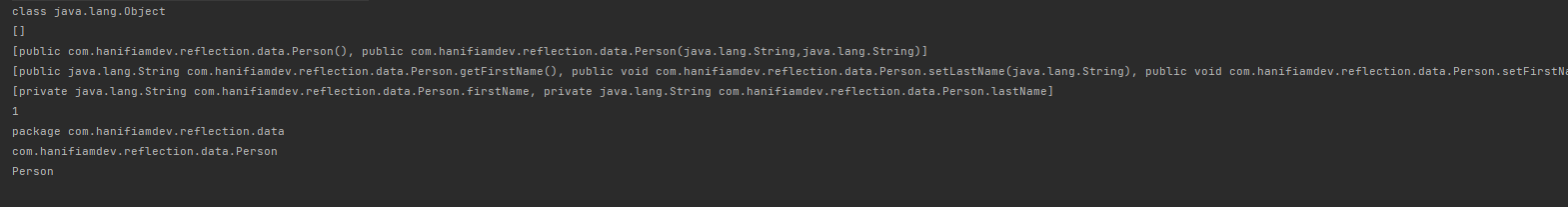
**Class<T> Method**

* Class<T> memiliki banyak sekali method
* Seperti untuk mendapatkan method, fields, constructor, annotation, superclass, interface, dan lain – lian
* Semua detail method nya bis akita baca di Javadoc
* <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Class.html>



Membuat unit test



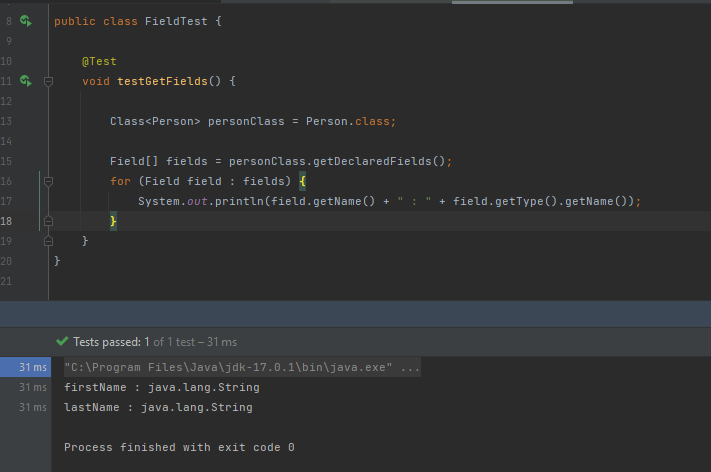


## **Field**

* 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, kitab isa menggunkan method getDeclaredFields()
* Atau kita juga bisa mendapatkan Field berdasarkan nama field nya menggunakan method

getField(name) atau getDeclaredField(name)

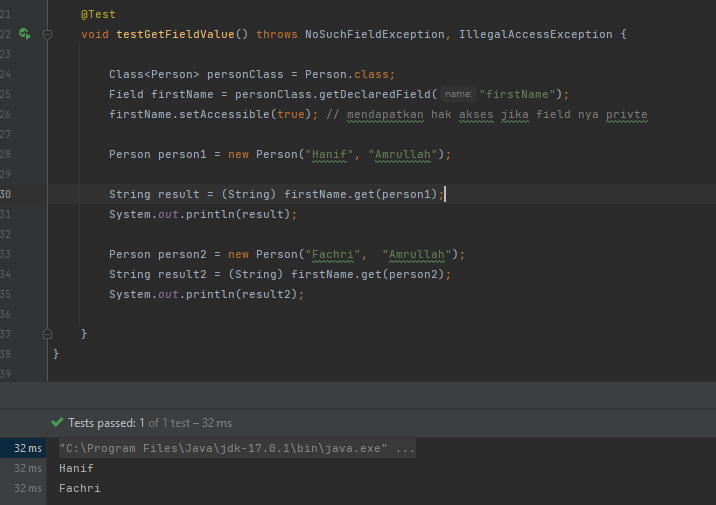
* Field sama seperti Class<T>, memiliki banyak sekali method yang bis akita gunakan untuk melihat detail dari Field tersebut, seperti tipe data, nama field, annotation, dan lain-lain
* <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/reflect/Field.html>



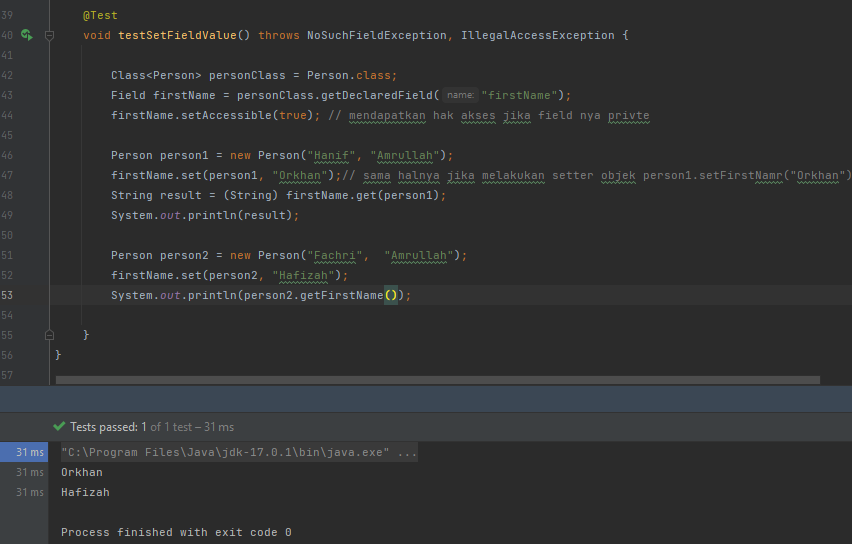
**Mengambil atau Mengubah Field Object**

* Field memiliki kemampuan untuk mengambil atau mengubah field dari object yang ada
* Misal kita sudah membuat Field, lalu kita memiliki object person 1, lalu kita ingin mengambil nilai Field tersebut atau mengubahnya, kita bisa menggunakan method setXxx() atau getXxx() pada Field

Mengambil Field dengan reflection



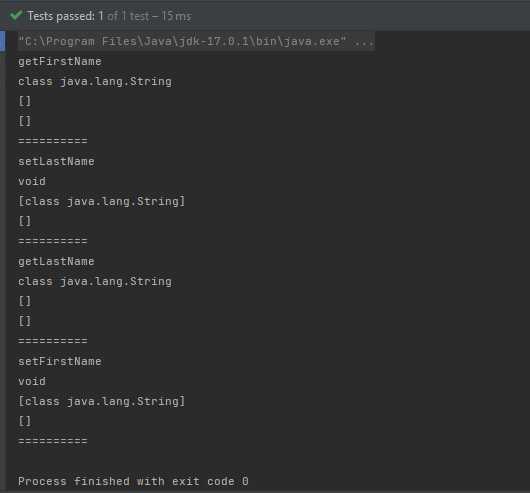
Mengubah Field dengan reflection



## **Method**

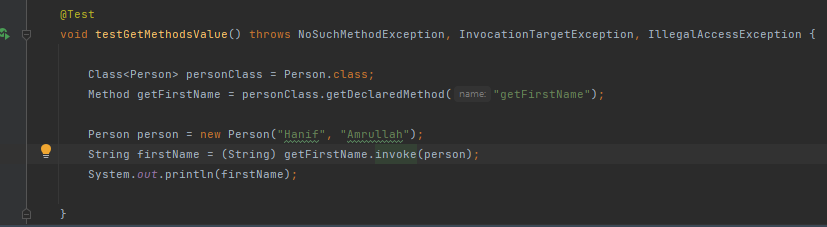
* Selain Field, kita juga bisa mendapatkan Method yang tersedia di Class<T>
* Cara mendapatkan pun sama seperti Field, kita bisa menggunakan method getMethods(), getDeclaredMethods(), getMethod(name), dan getDeclaredMethod(name)
* Method pun banyak sekali method yang bis akita gunakan untuk mendapatkan 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>





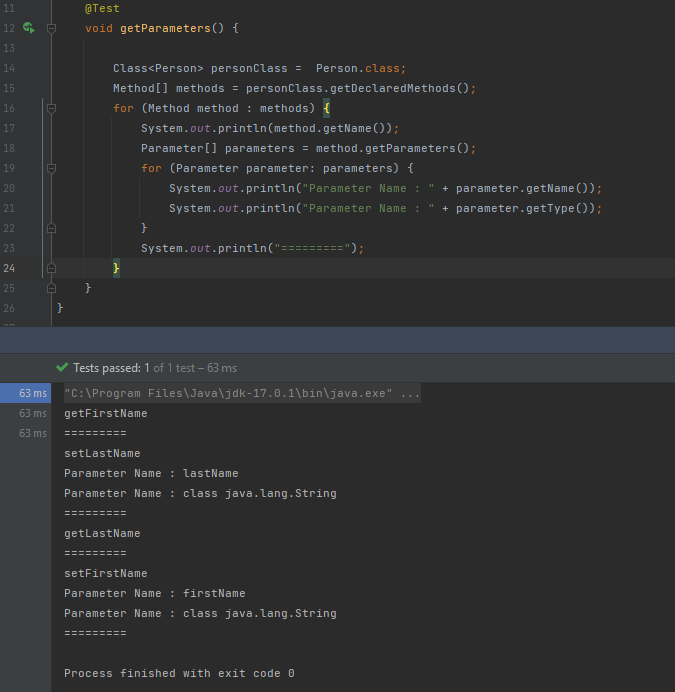
**Memanggil Method Object**

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



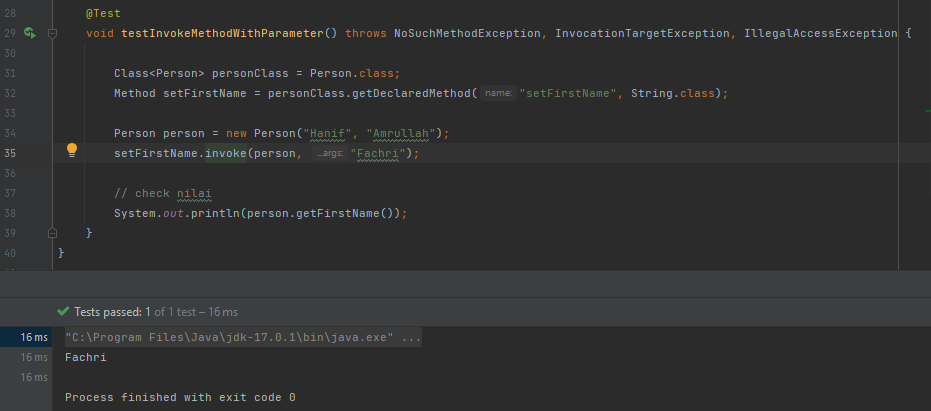
## **Parameter**

* 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>



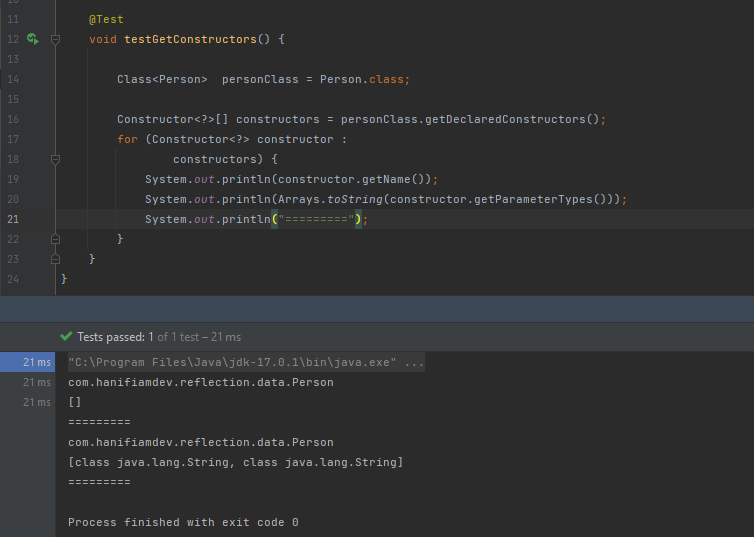
**Memanggil Method Object dengan Parameter**

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



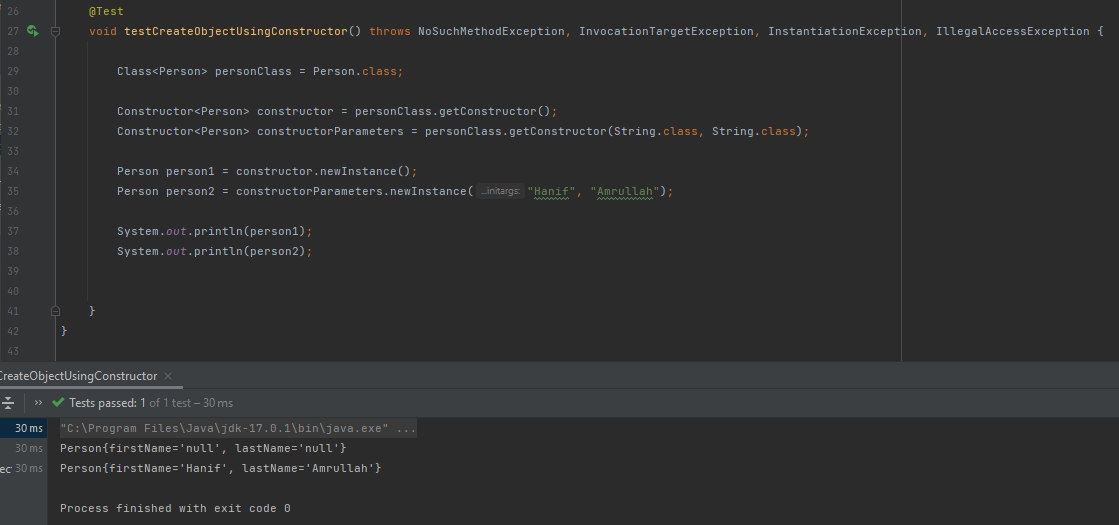
## **Constructor<T>**

* 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>



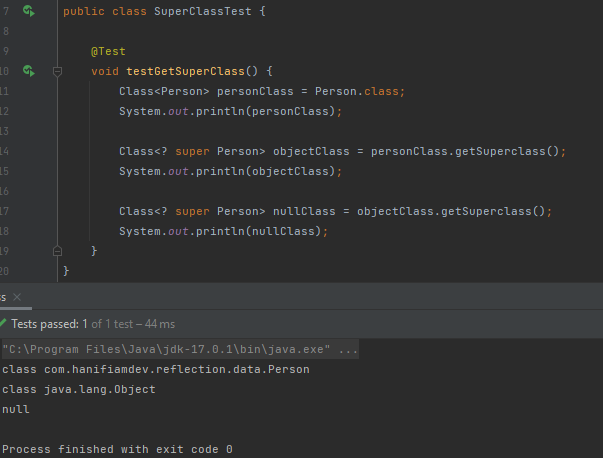
**Membuat Object dengan Constructor**

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



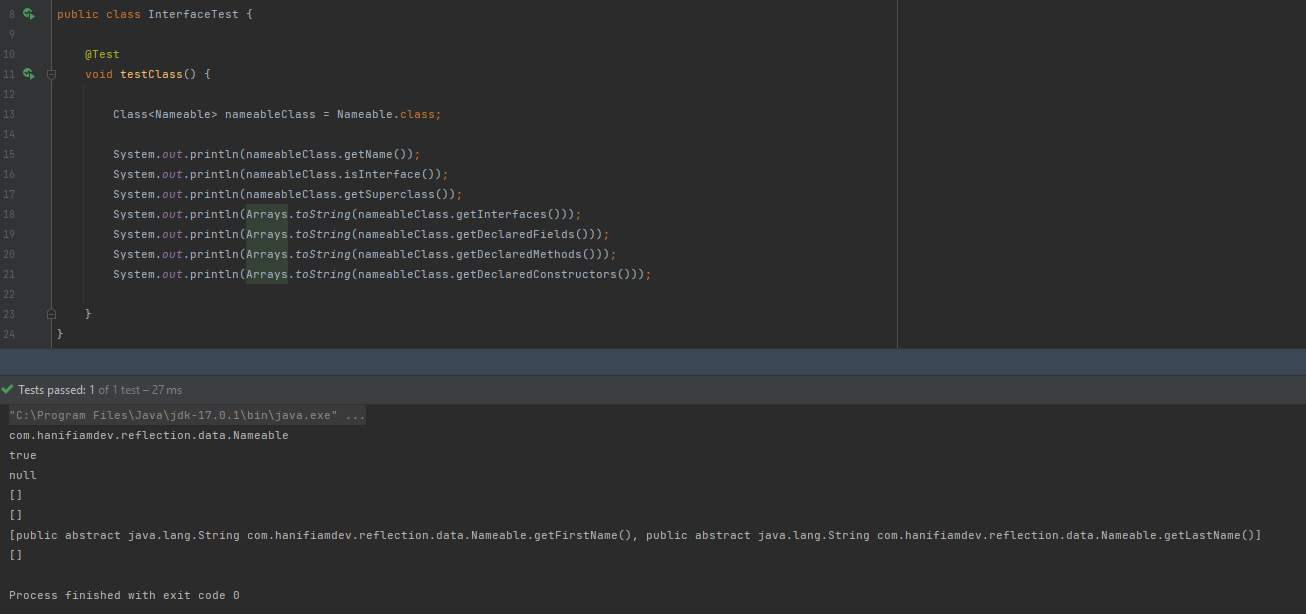
## **Super Class**

* 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()>



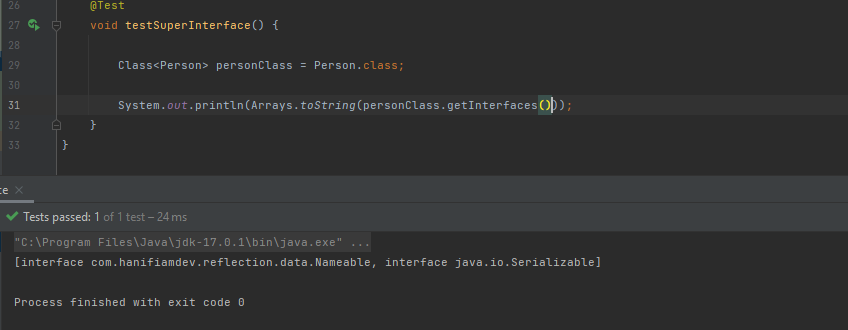
## **Interface**

* 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()



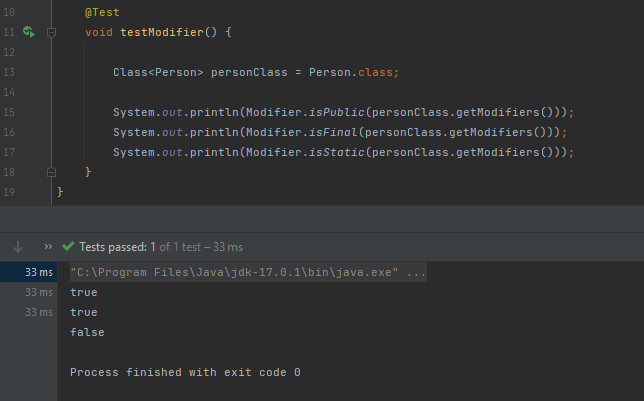
**Super Interface**

* 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()



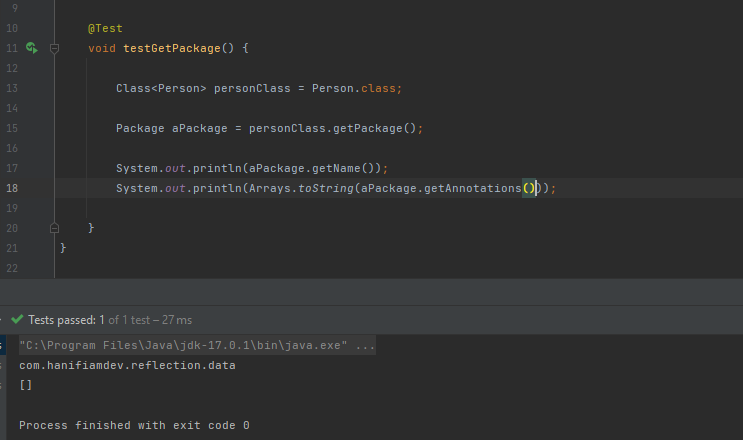
## **Modifier**

* 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, kitab isa 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>



## **Package**

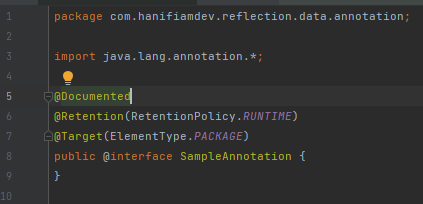
* Package merupakan representasi dari Java Package
* Kita bisa mendapatkan Package dari Class<T> dengan menggunakan method getPackage()
* Terdapat banyak informasi di dalam Package yang bisa kita gunakan
* <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Package.html>



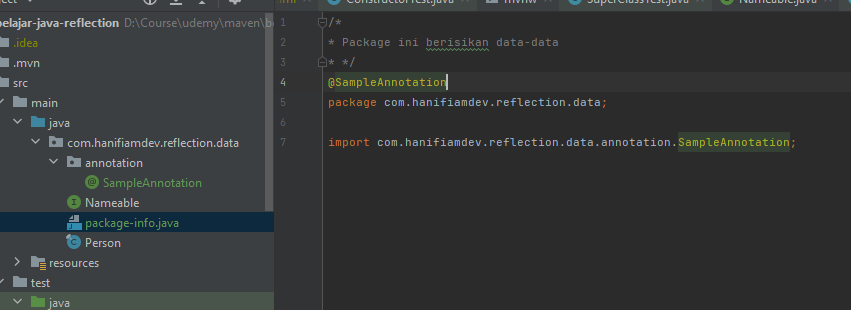
**Package-info.java**

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

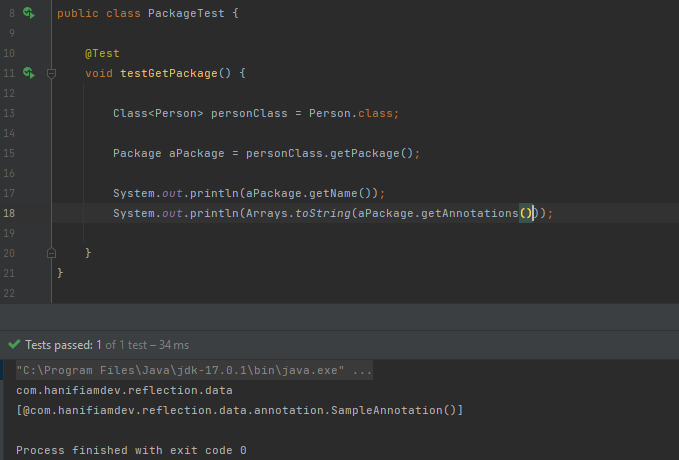
Membuat annotaion



Include kedalam package-info.java



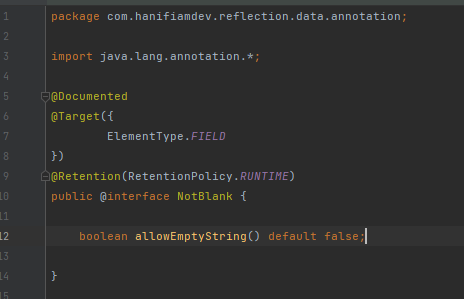
Running unit test kemabali



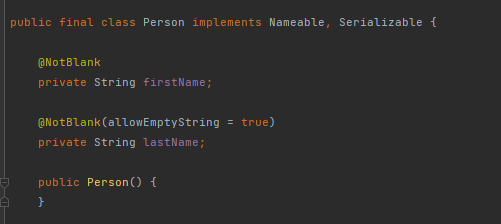
## **Annotation**

* 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
* <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Annotation.html>

Buat annotation dengan nilai default false artinya tidadk boleh blank

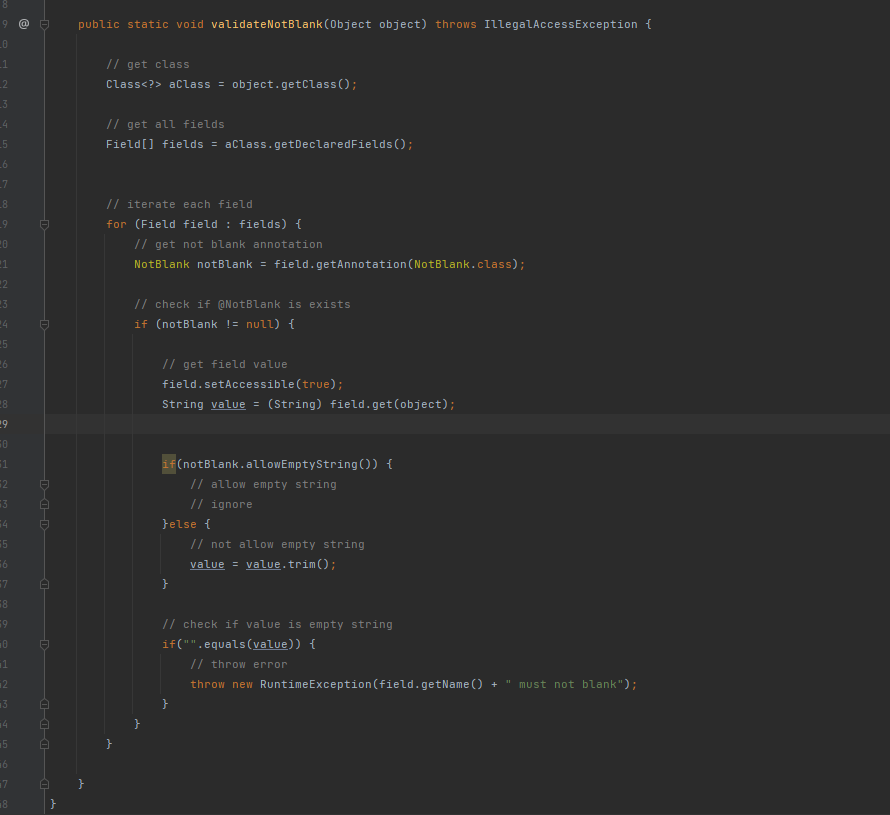


Masukkan annotation ke kelas Person

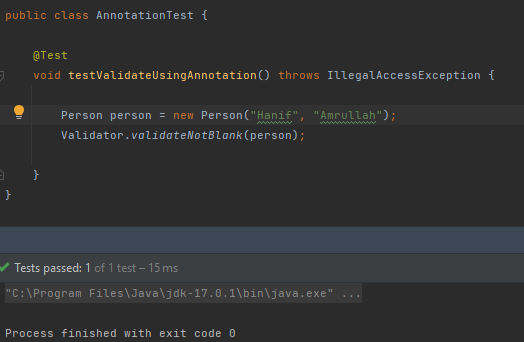


Note : field firstName tidak boleh Blank(null maupun empty ) namun lastName dibolehkan karena alloEmptyString di set true

Buat class validator dengan method static



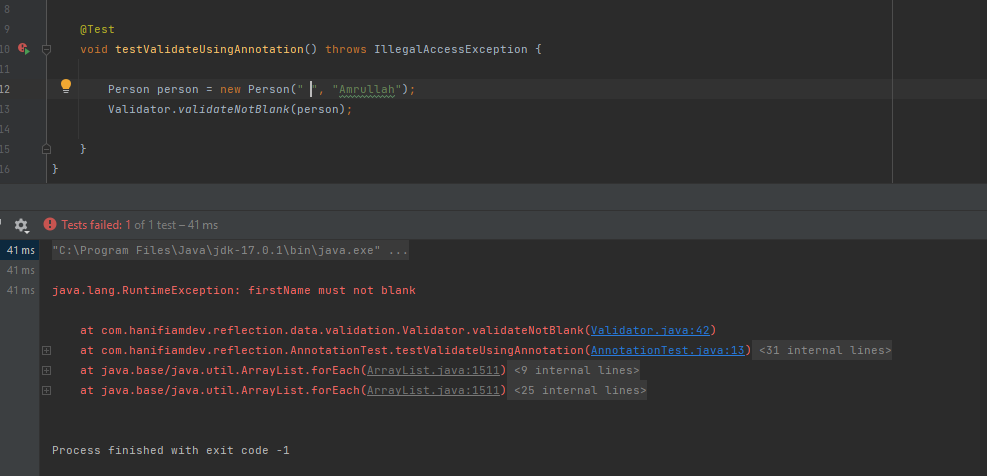
Selanjutnya kita test case 1 (firstname dan lastname diisi semua)



Case ke 2 (lastname coba di isi dibut blsnk dan hasilnya sukses



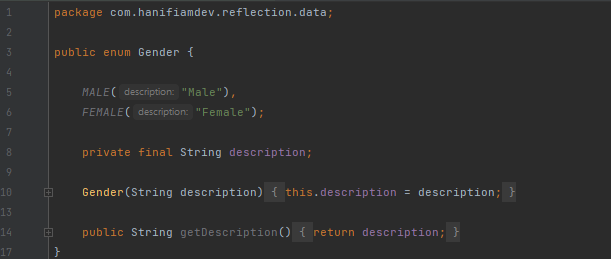
Case ke 3 (firsname coba dibuat blask) maka akan kena eror



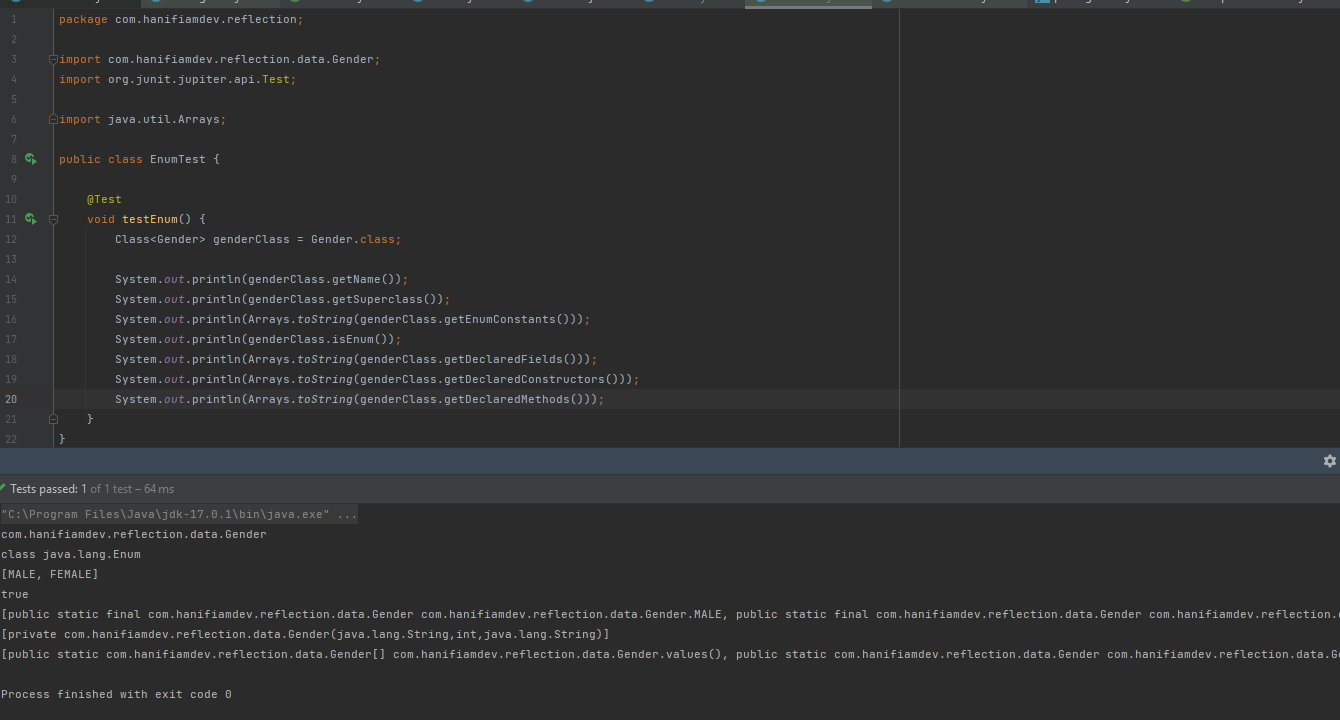
## **Enum**

* Enum di Java Reflection, sama seperti Interface, direpresentasikan dengan Class<T>, hal ini dikarenakan, pada Enum juag 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>

Membuat Kelas enum

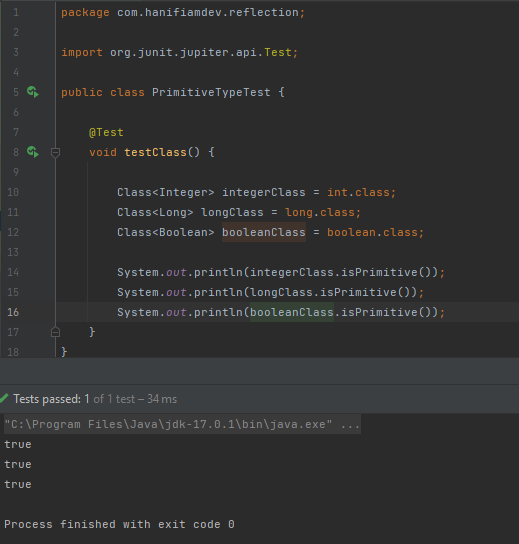


Test melakukan reflection enum



## **Primitive Type**

* Apa yang terjadi jika 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 mmembuat 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



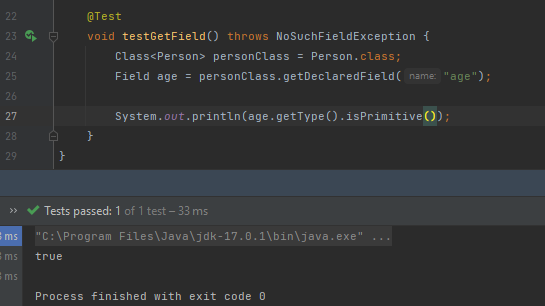
**Mengambil Data Primitive Type**

* 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

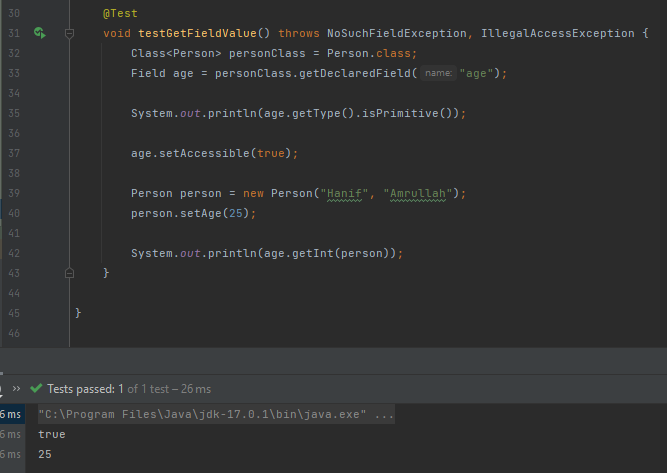
Misal menambahkan field age di dalam kelas Person



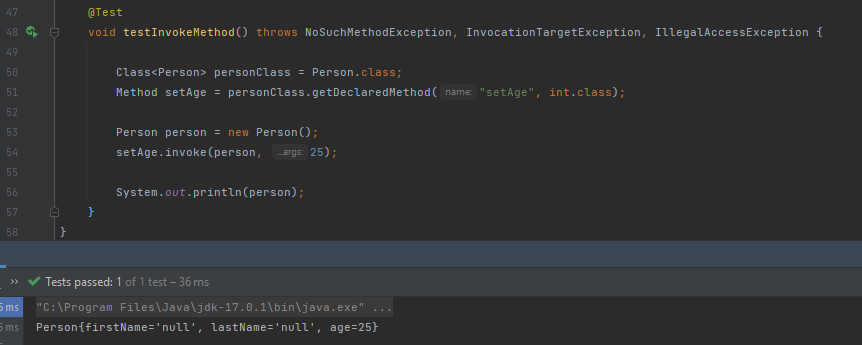
Check apakah field merupakan type primitive



Mengambil field data primitive

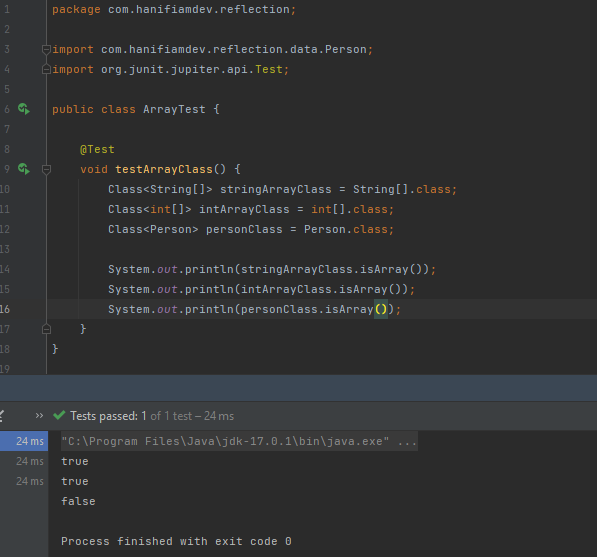


Melakukan eksekusi method primitive type



## **Array**

* Sama seperti tipe data yang lainnya, Arraypun di representasikan dalam bentuk Class<T> di Java Reflection
* Untuk membuat Class<T> Array, kita bisa gunakan .class 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



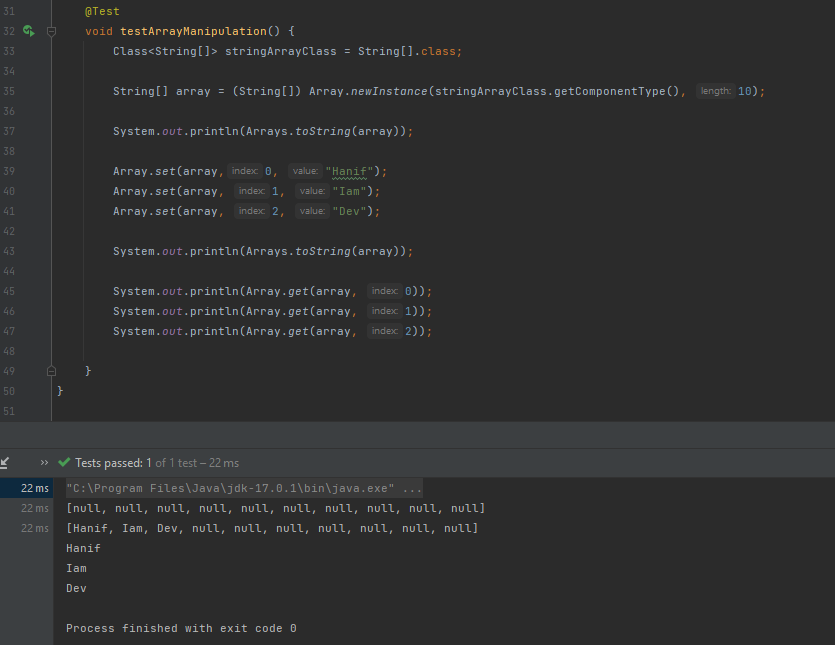
**Array Member**

* 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(), get Methods, getConstructor(), maka hasilnya adalah kosong



**Java.lang.reflect.Array**

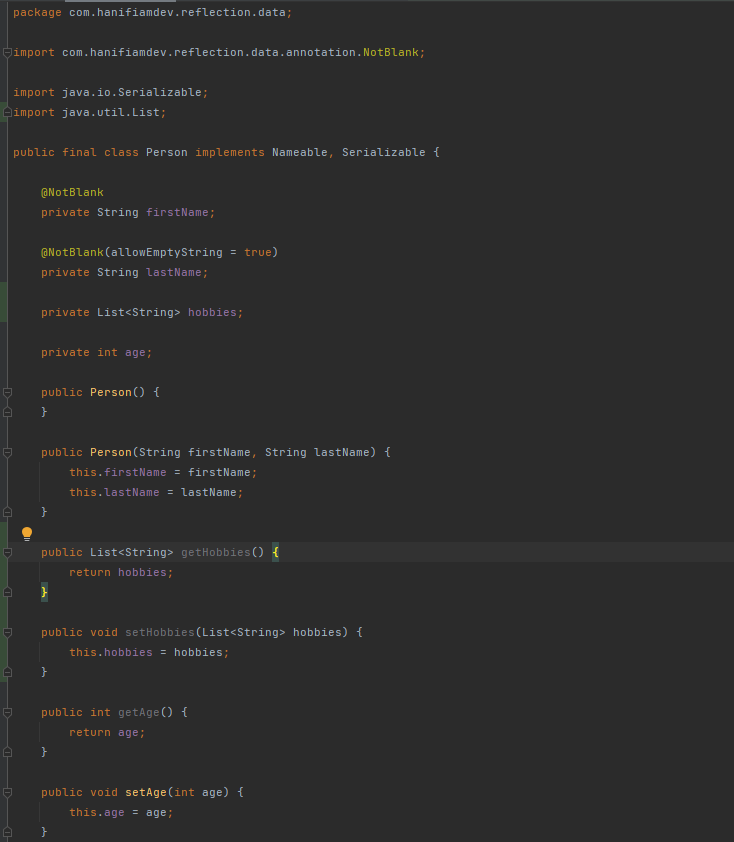
* Lantas bagaimana jika kita ingin mengakses data array, membuat array dan lain – lain, jika pada Class<T>, semua class member nya tidak tersedia di Array?
* Untungnya terdapat class java.lang.reflect.Array, yang bisa kita gunakan untuk membantu menggunakan Class<T> dengan tipe Array
* Ada banyak method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



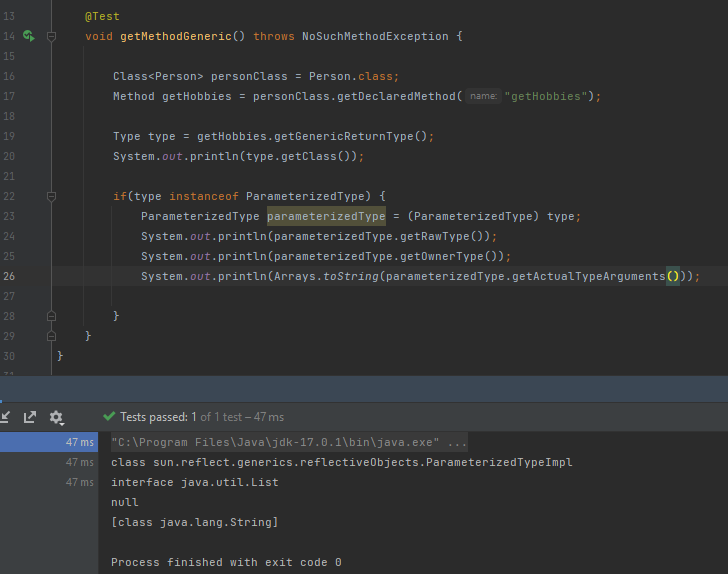
## **Prameterized Type**

* 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>

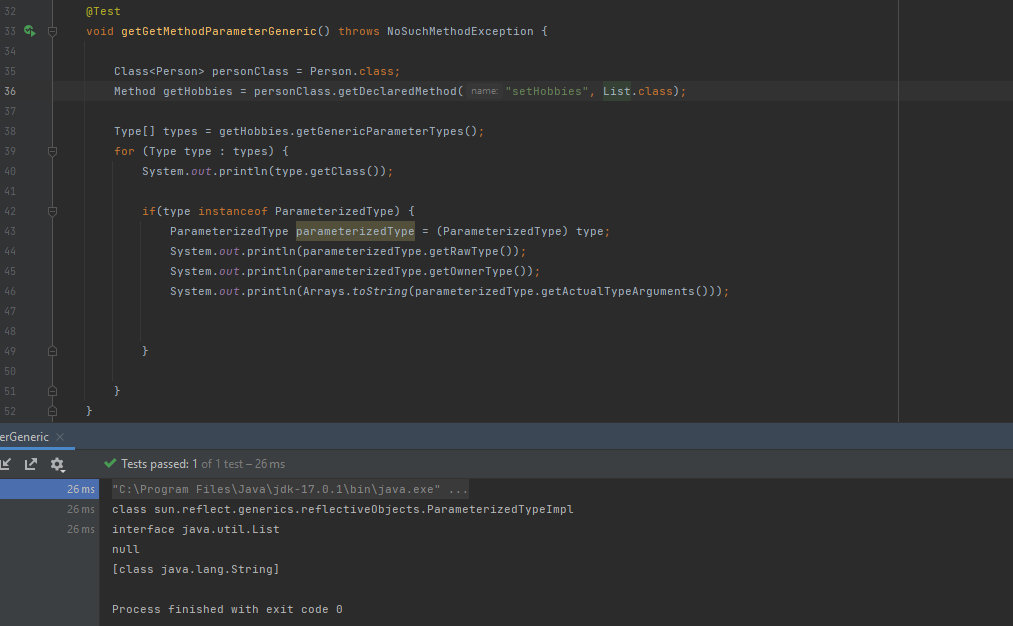
Menambah Generic Field



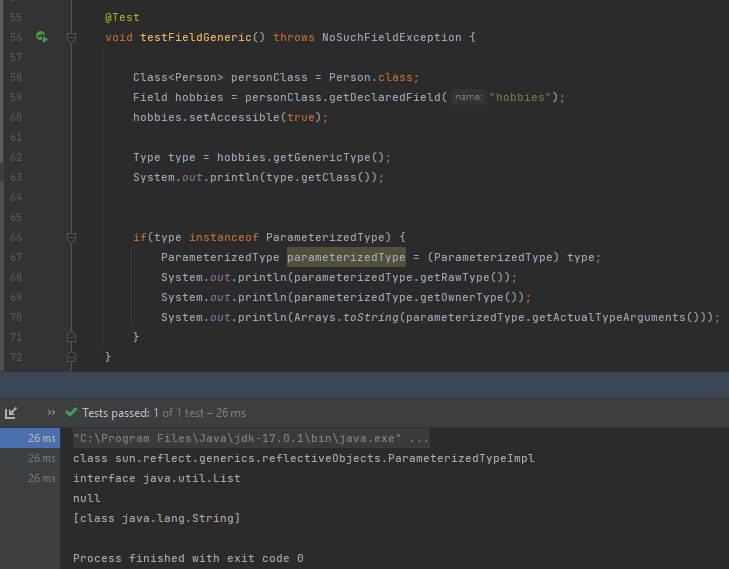
Parameterized Return Type



Parameterized Parameter Type



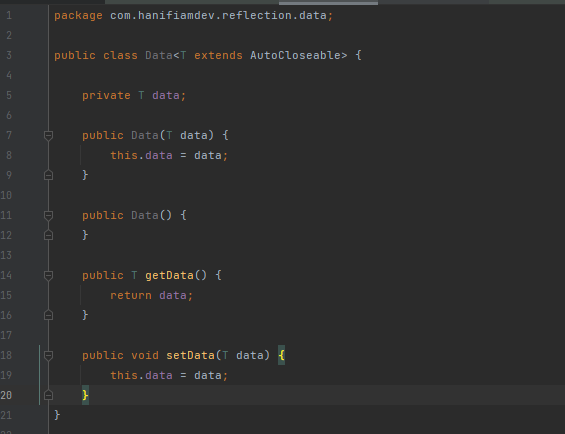
Parameterized Field



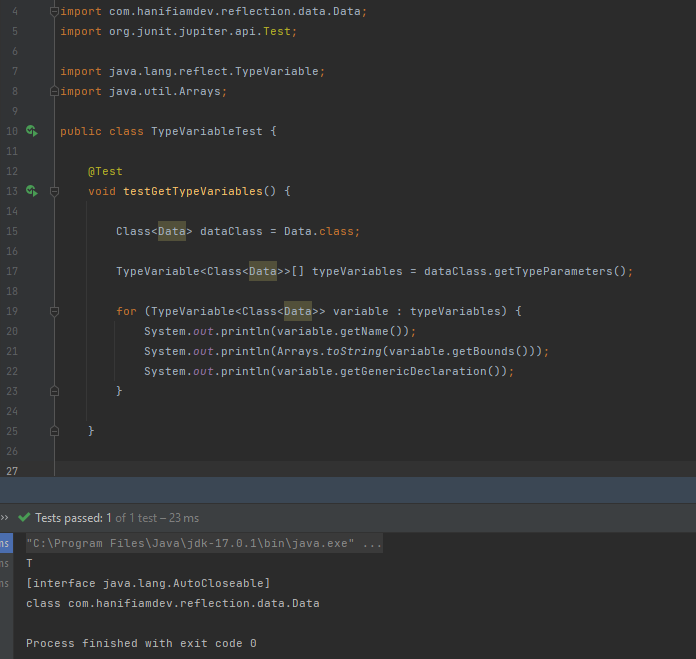
**Type Variable**

* 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>

Buat Generic Class



Type Variable di Class



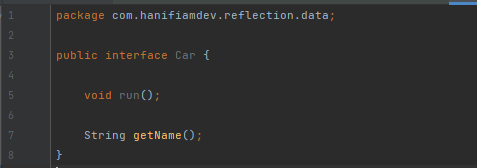
## **Proxy**

* Saat kita belajar di kelas Java Unit Test, kita belajar dengan 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>

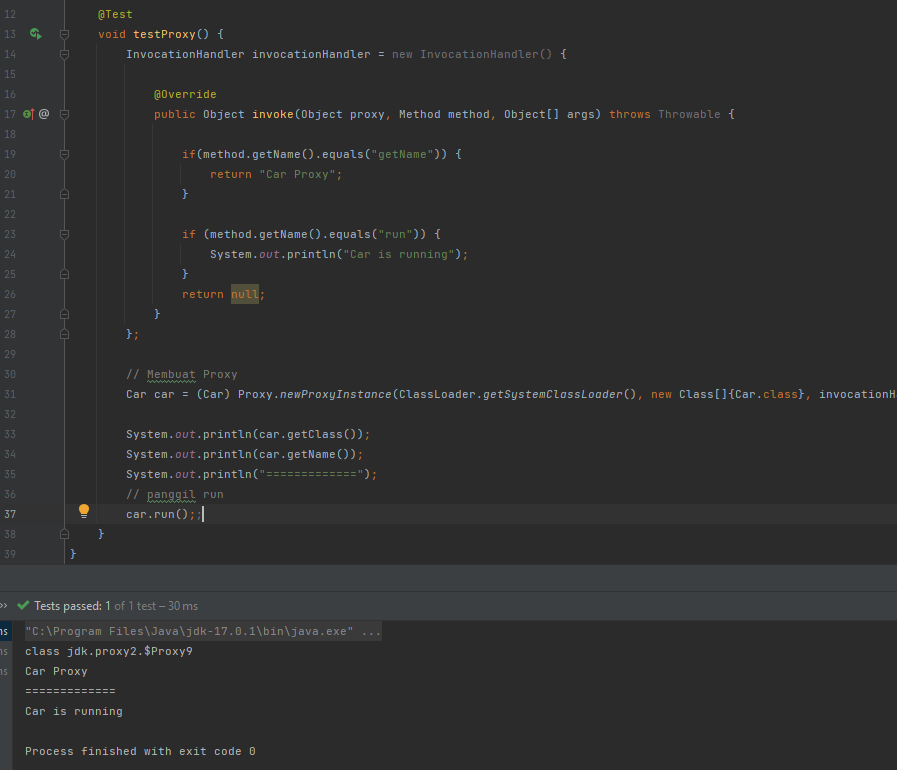
**Invocation Handler**

* Saat kita membuat proxy, kitab isa secara dinamis menerima method yang dipanggil, dan mengembalikan value di method tersebut denga 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>

Membuat Interface



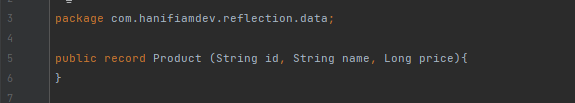
Membuat Invocation Handler



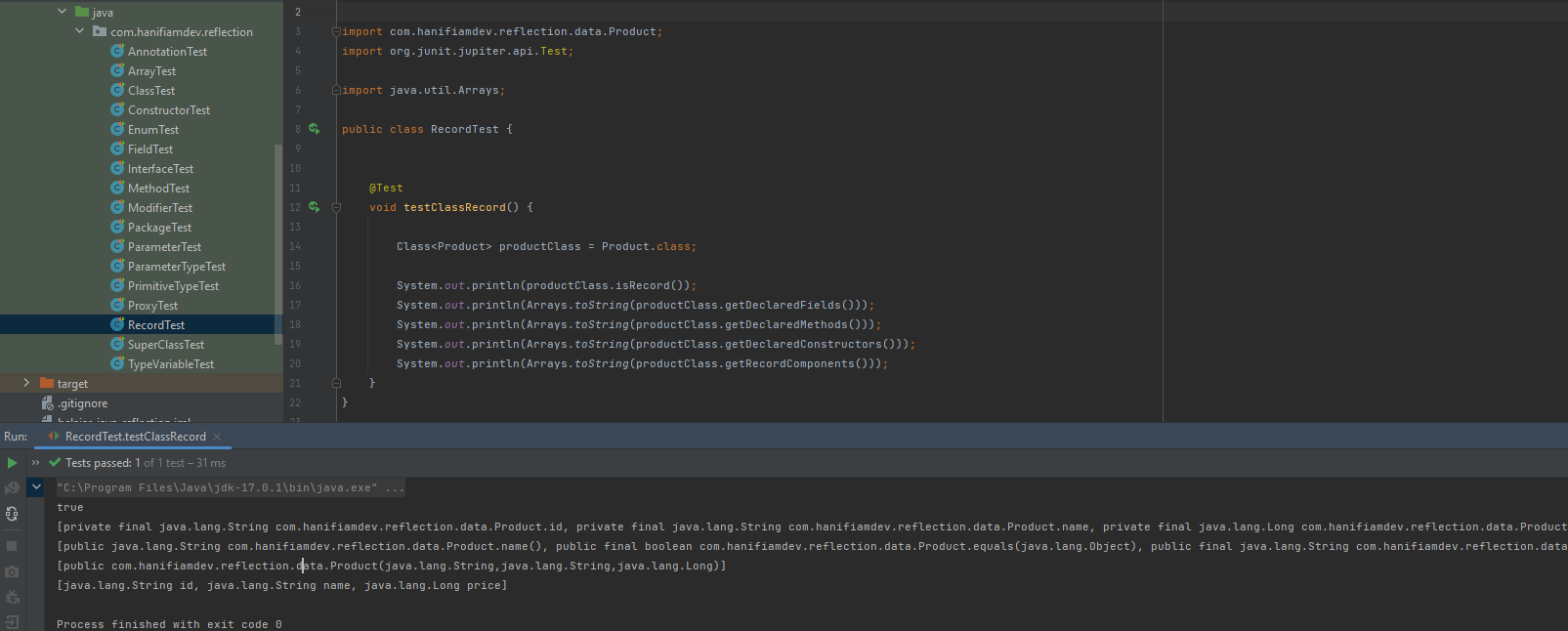
## **Record**

* Di Java versi 16, fitur Java Record sudah stabil dan kita juga bisa menggunakan java Reflection
* Record di Java Reflection tetap di representasikan dengan Class<T>
* Hanya saja method isRecord() akan mengembalikan nilai true, selain itu untuk mendapatkan detail Record, kitab isa 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/11/docs/api/java.base/java/lang/reflect/RecordComponent.html>

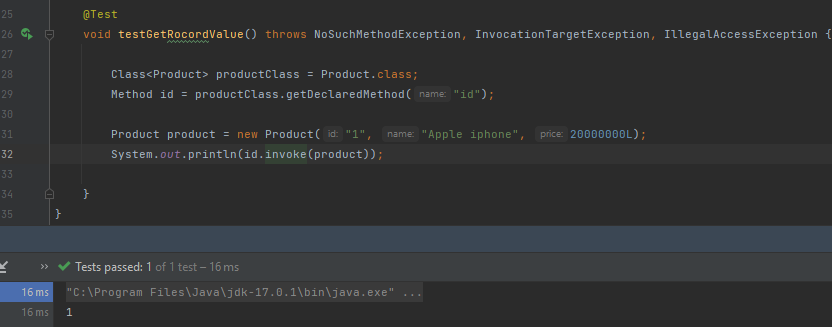
Membuat Record



Test



Mengaubah Record Component



Get Record dengan Accessor

