**Java Validation**

## **Pendahuluan**

**Sebelumnya harus sudah paham materi terkait** :

* Java Object Oriented Programming
* Java Generic
* Java Collection
* Apache Maven
* Java Date & Time API
* Java Unit Test
* Java Reflection

**Agenda**

* Pengenalan Bean Validation
* Validator
* Constraint
* Constraint Violation
* Message Interpolation
* Grouping
* Custom Validation
* Metadata
* Dan lain-lain

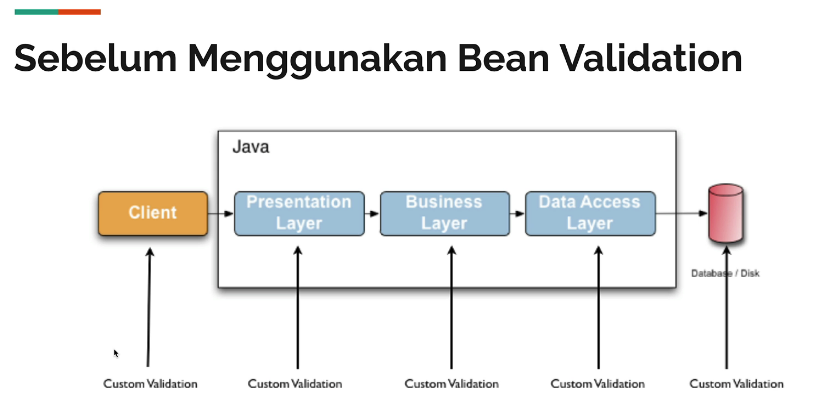
## **Pengenalan Bean Validation**

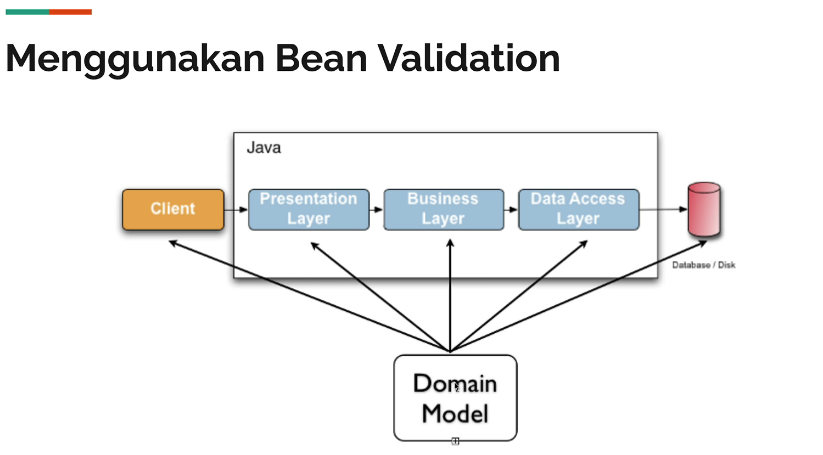
**Bean Validation**

* Bean Validation merupakan standard spesifikasi untuk melakukan validation di Java
* Bean Validation sendiri merupakan spesifikasi yang masuk ke dalam teknologi Java Enterprise / Jakarta Enterprise
* Dengan Bean Validation, kita bisa membuat kode validation yang standard dijava
* <https://jakarta.ee/specifications/bean-validation/>

**Hibernate Validator**

* Bean Validation hanyalah spesifikasi standard isinya hanyalah kumpulan kontrak Interface dan Annotation, mirip seperti JDBC ( Java Database Connectivity )
* Untuk implementasinya, kita butuh Driver untuk Bean Validation
* Hibernate Validator adalah salah satu driver atau implementasi Bean Validation yang populer saat ini
* <https://hibernate.org/validator/>





**Jakarta Bean Validation**

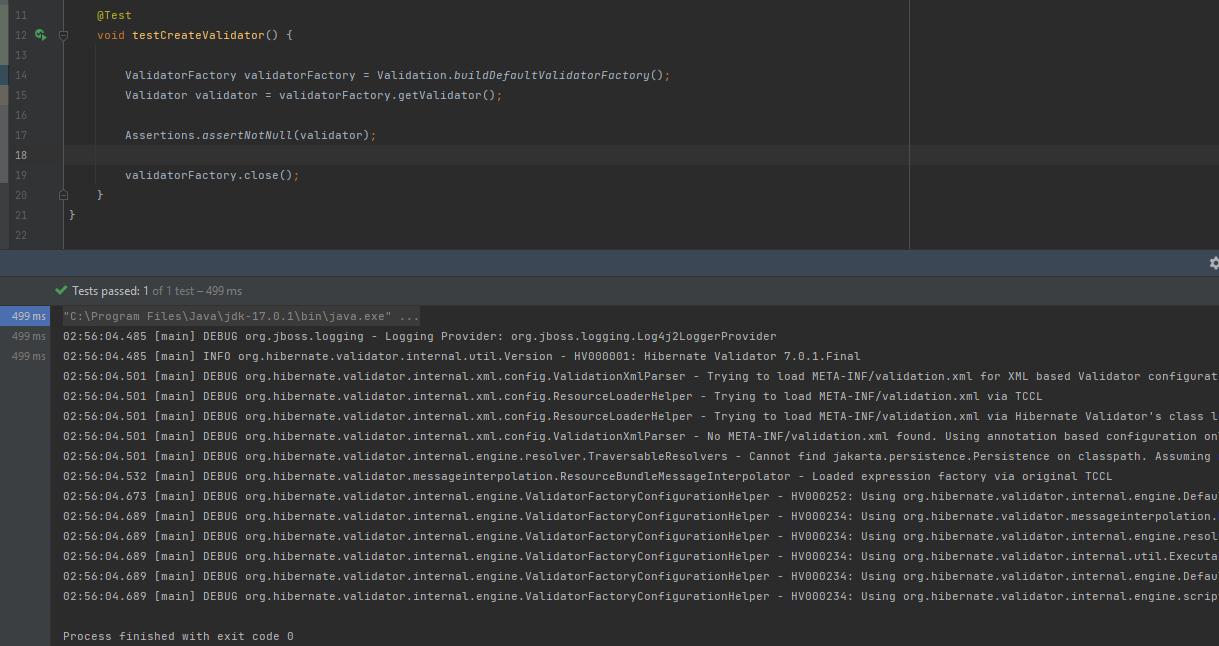
* Saat ini versi terbaru dari Bean validation adalah versi 3, sedangkan masih banyak yang menggunakan Bean Validation versi 2
* Bean Validation versi 2 masih menggunakan nama Java Enterprise, sedangkan Bean Validation versi 3 sudah menggunakan nama Jakarta
* Oleh karena itu saat ini, banyak package untuk Bean Validation sudah berubah, dari yang sebelumnya menggunakan package javax.validation menjadi Jakarta.validation
* Oleh karean itu jangan bingung jika masih menggunakan Bean Validation versi 2, tinggal diubah nama package nya saja

## **Validator**

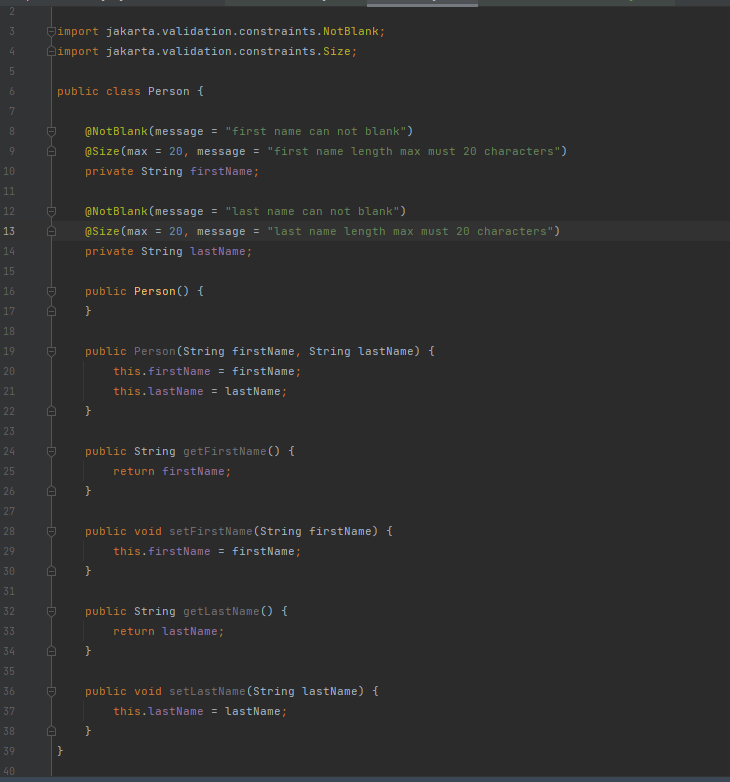
* Validator adalah class utama dalam Bean Validation
* Validator digunakan sebagai object untuk mengeksekusi validation
* Validator adalah object yang berat, oleh karena itu sebaiknya hanya dibuat satu kali saja dalam aplikasi
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/validator>

**ValidatorFactory**

* Validator merupakan sebuah interface, untuk membuatnya kita butuh bantuan object ValidatorFactory
* ValidatorFactory merupakan sebuah interface yang digunakan untuk membuat object-object yang ada di Bean Validation
* Salah satu method nya adalah getValidator(), yang digunakan untuk mendapatkan object Validator
* ValidatorFactory cukup dibuat sekali di setiap aplikasi, karena object ini merupakan object Validator berat, dan untuk membuatnya kitab isa menggunakan class Validation dan method buildDefaultValidatorFactory()



Kode : Class denga Constraint



## **Constraint**

**Manual Validation**

* Sebelum menggunakan Bean Validation, untuk melakukan validasi di Java, biasanya kita lakukan secara manual
* Biasanya kita menggunakan if else untuk melakukan pengecekan
* Dan yang terjadi validasi error, biasanya kita akan buat exception terjadi
* Pada Bean Validation, Cara kerjanya berbeda, kita tidak butuh melakukan validasi secara manual lagi, validasi di Bean Validation menggunakan Annotation yang bisa kita simpan pada Field, Method, Parameter dan lain-lain

**Constraint**

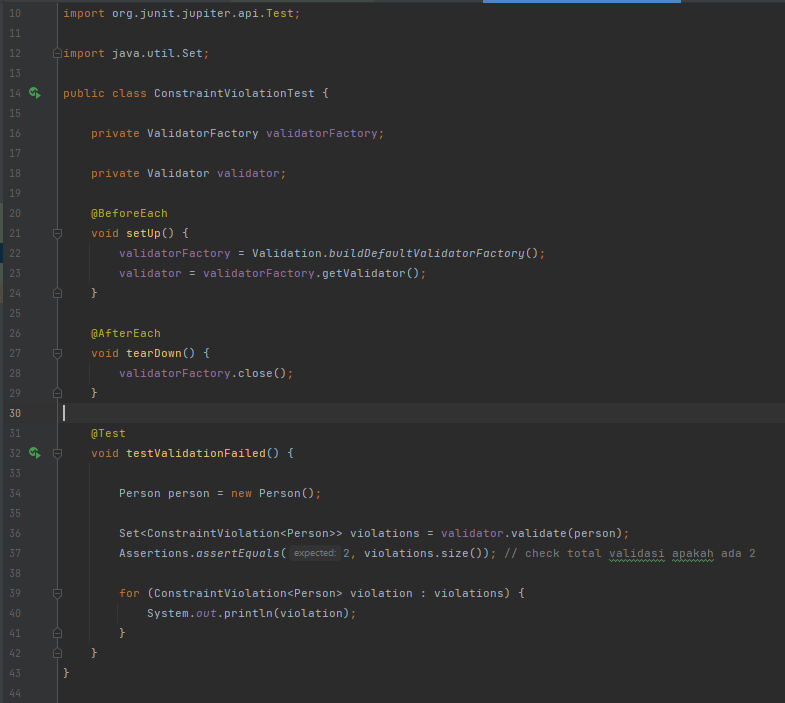
* Constraint merupakan Annotation yang digunakan sebagai penanda untuk target yang kita tambahkan (misal Field, Method, dan lain-lain)
* Bean Validation sudah menyediakan banayak sekali Constraint yang bisa langsung kita gunakan
* Jika kita butuh validasi yang berbeda, kita juga bisa membuat constraint secara manual, yang akan kita bahas nanti di chapter tersendiri
* Semua Constraint di Bean Validation terdapat di package Jakarta.validation.constrains
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraints/package-summary.html>

## **Constraint Validator**

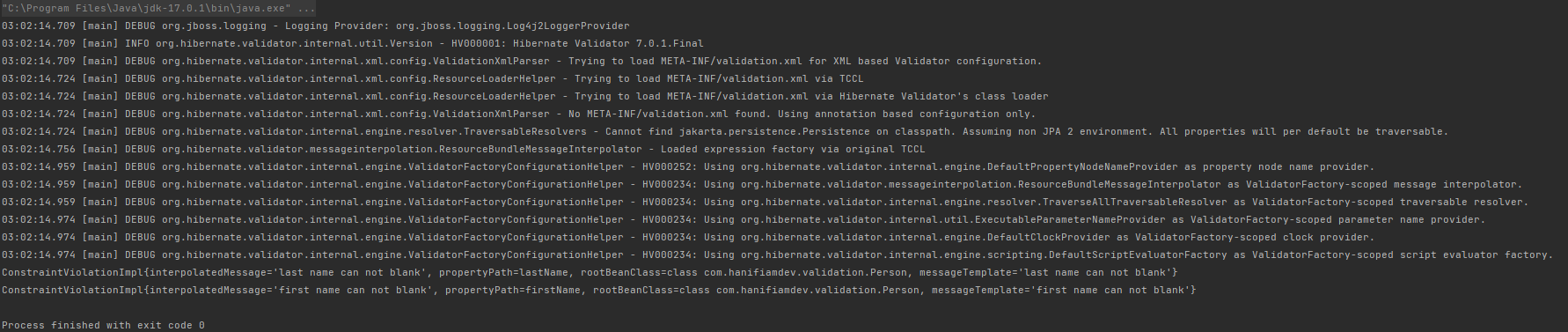
**Constraint Violation**

* Setelah kita menambahkan annotation Constraint ke class yang akan kita validasi, selanjutnya kita bisa mulai melakukan validasi terhadap object class tersebut menggunakan method validate() milik class Validator
* Hasil kembalian dari method validate() adalah Set<ConstraintViolation>, dimana ConstraintViolation tersebut merupakan representasi kesalahan dari constraint
* Jika terdapat kesalahan, otomatis terdapat ConstraintViolation, namun jika tidak ada kesalahan, maka tidak akan terdapat ConstraintViolation, alias Set nya akan berisi data kosong
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraintviolation>

Kode : Constraint Violation



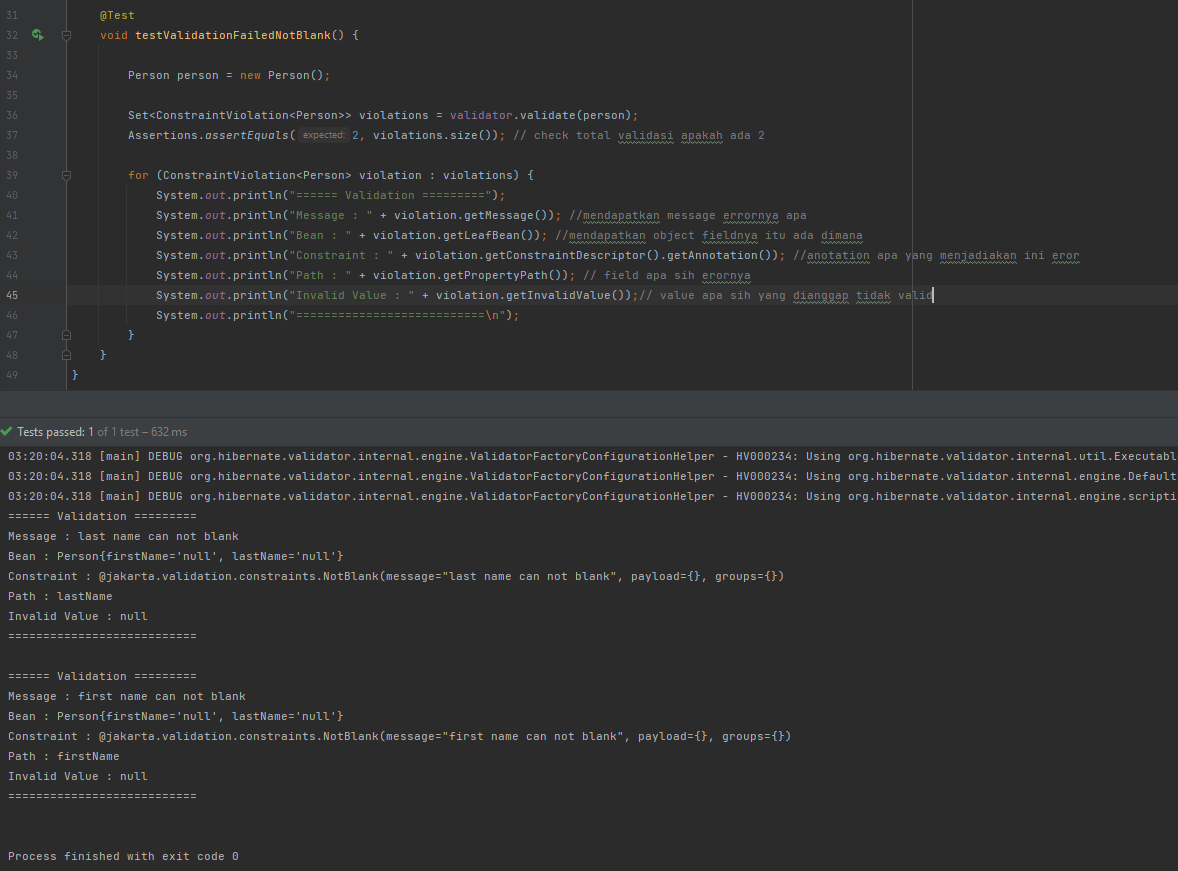
Result : Constraint Violation



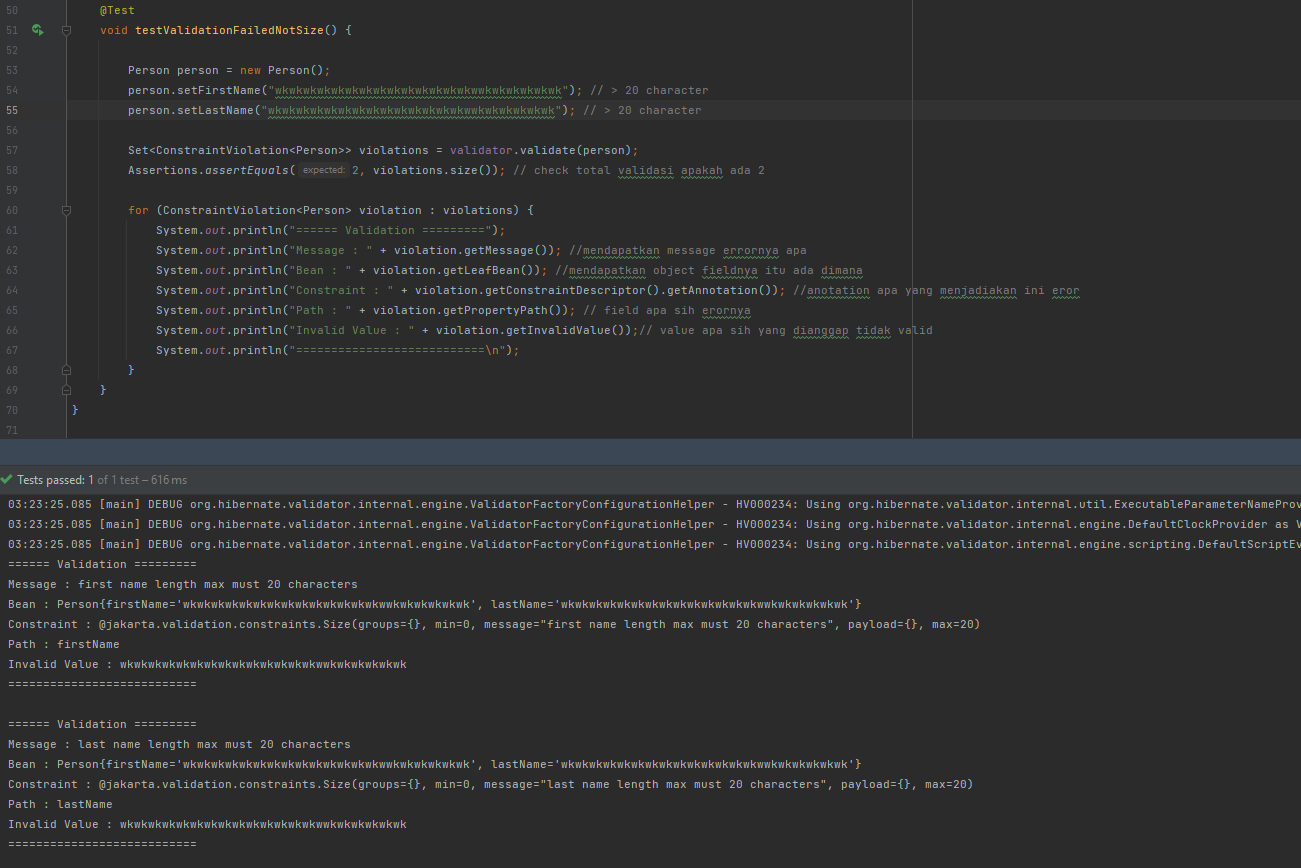
**Object Metadata**

* Jika kita perhatikan, pada Constraint Violation, tidak hanya message error yang bisa kita lihat, kita juga bisa melihat field mana yang error, dari object mana, dan lain-lain
* Ini sangat bagus ketika kita ingin melakukan debugging field mana yang error

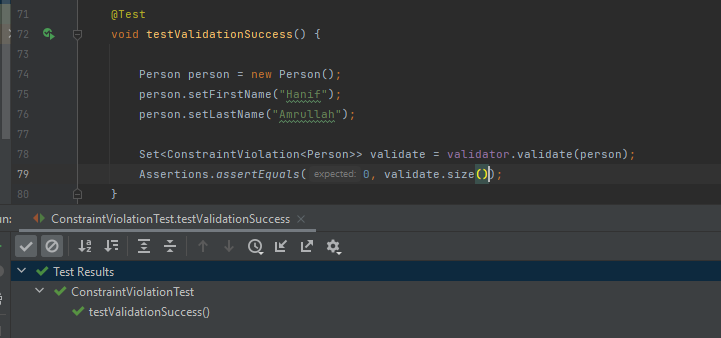
Test validasi Not Blank



Test validasi Size



Test Sukses validasi



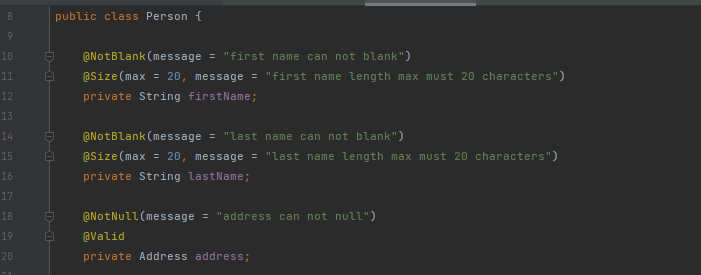
## **Nested Validation**

* Secara default, jika terdapat nested object, Bean Validation tidak akan melakukan validasi terhadap data object tersebut
* Misal kita punya class Person, dimana memiliki field address dengan tipe class Address, secara default isi dalam class Address tidak akan divalidasi
* Jika kita ingin melakukan validasi terhadap nested object tersebut, kita perlu menambahkan annotation @Valid
* @Valid juga bisa digunakan untuk nested object yang terdapat di dalam Array atau Collection
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/valid>

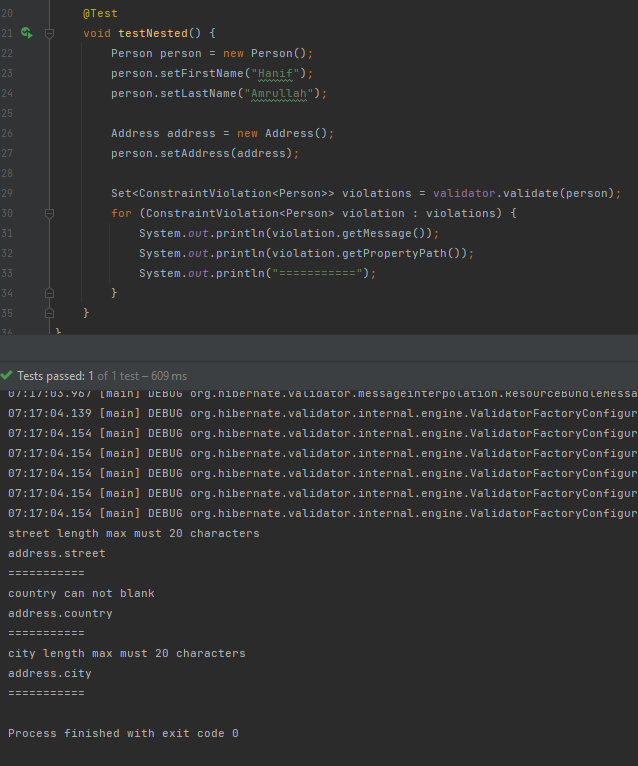
kode : Class Address



Add field address beserta getter and setter di Person(@Valid tujuannya supaya ikut divalidasi)



Address ikut divalidasi karena di kelas person sudah ditambahkan @Valid



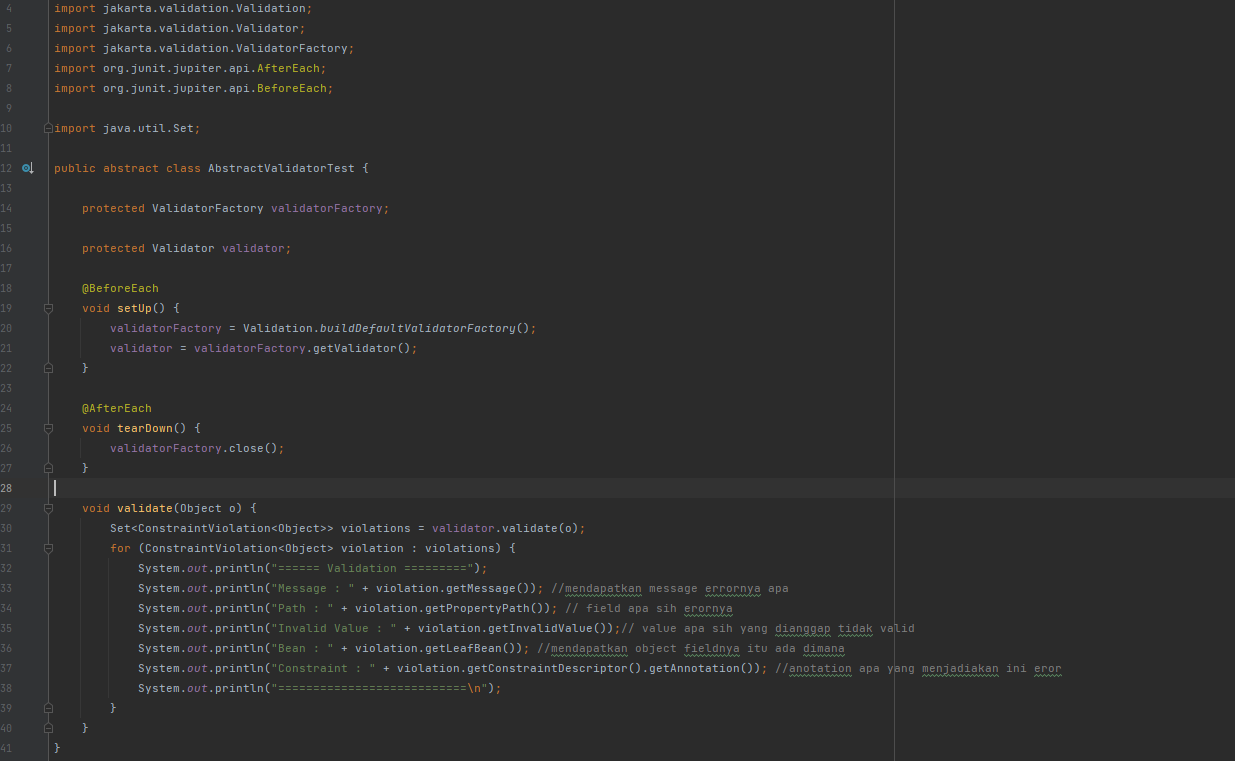
## **Hibrenate Validator Constraint**

* Selain Annotation Constraint yang terdapat di Bean Validation
* Hibernate Validator juga menyediakan Constraint tambahan
* Kita bisa melihat nya di package org.hibernate.validator.constraints
* <https://docs.jboss.org/hibernate/stable/validator/api/org/hibernate/validator/constraints/package-summary.html>

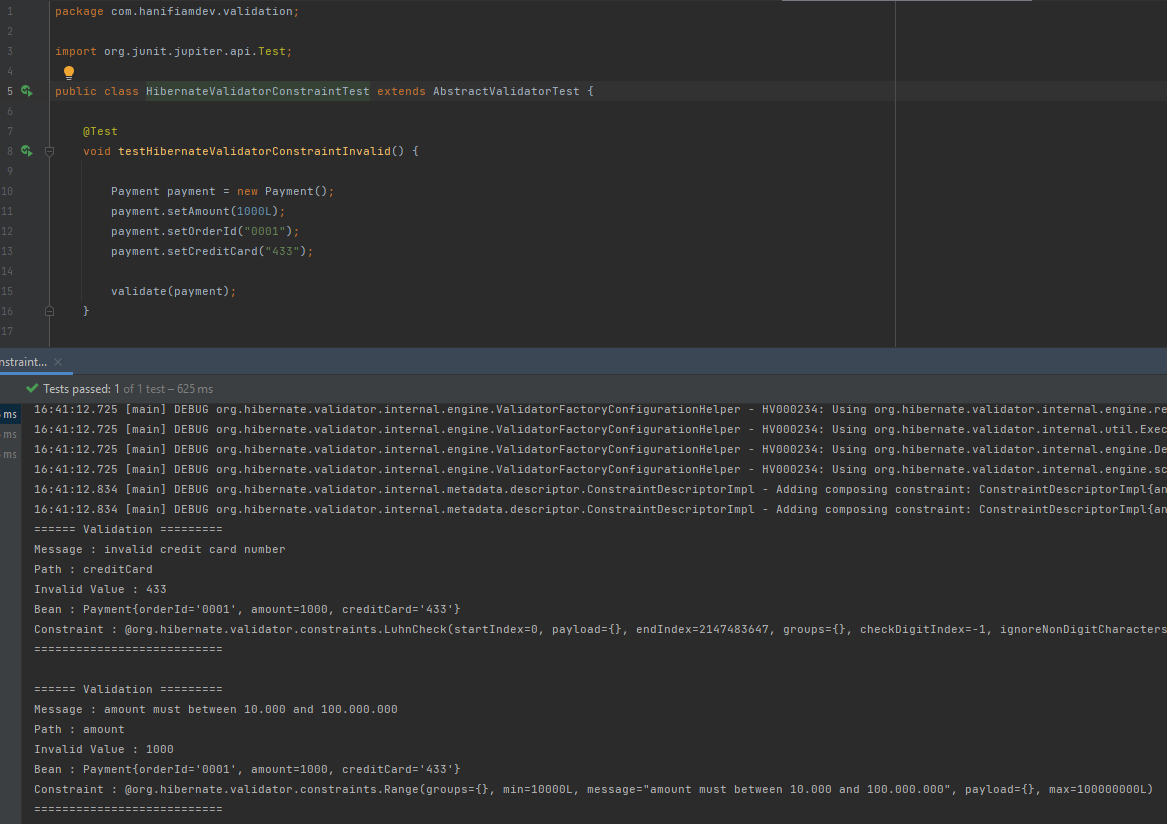
Payment Class



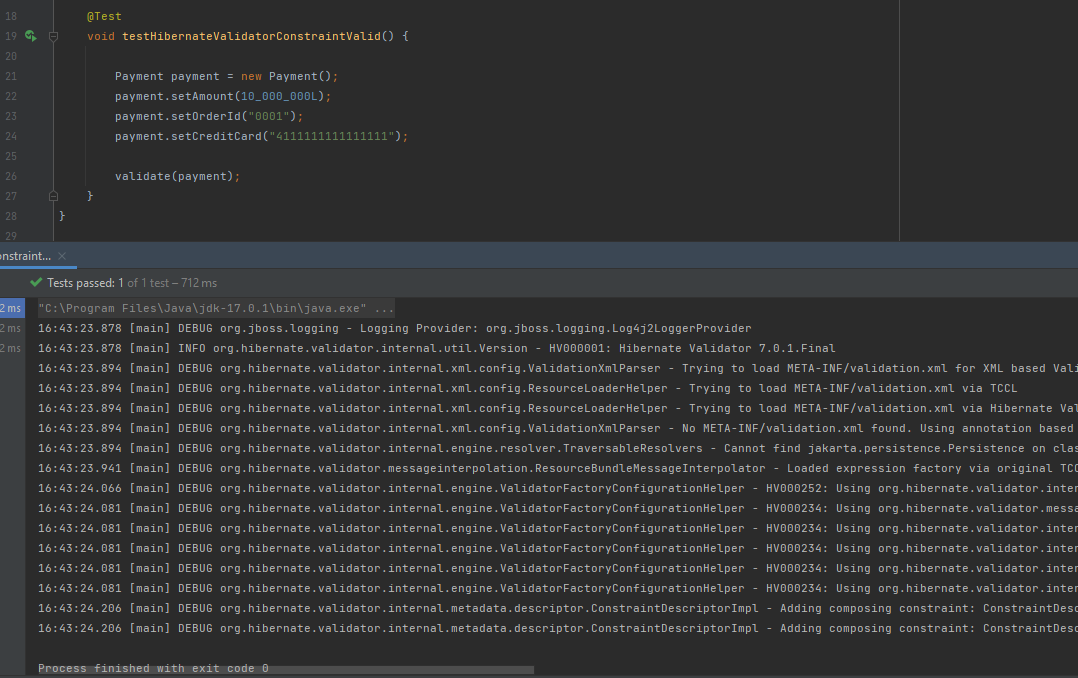
Buat kelas abstract supaya inisialisasi validator lebih mudah dan reusable



Test Invalid



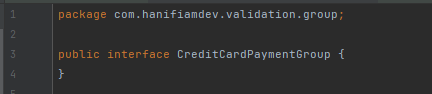
Test Valid

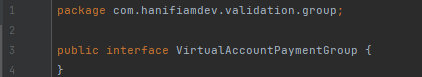


**Group Constraint**

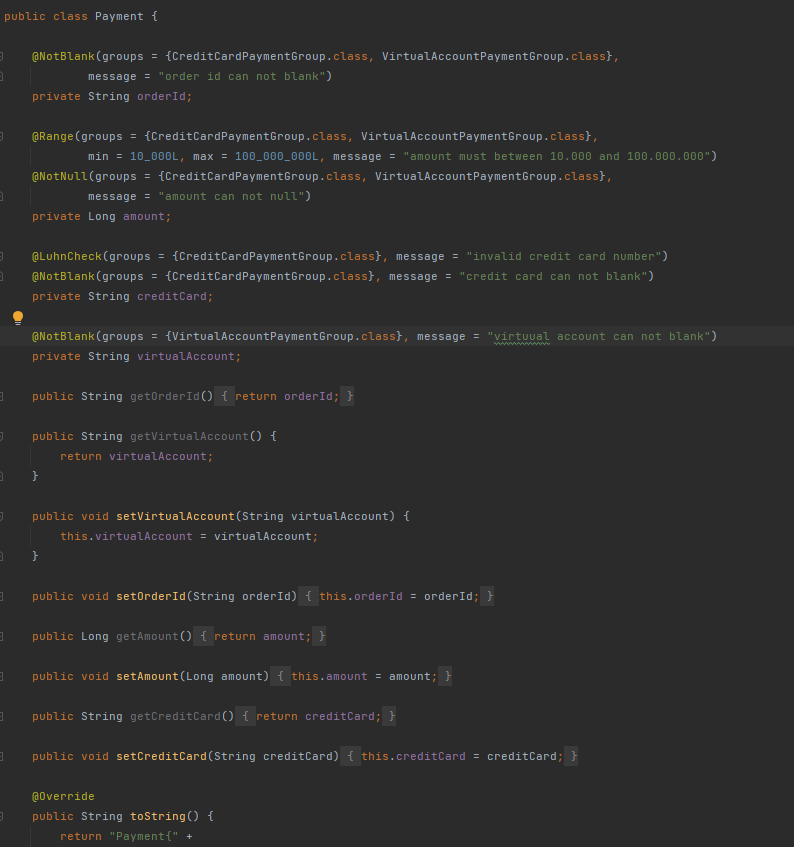
* Secara default, saat Validator melakukan validasi, validator akan memvalidasi semua field yang memiliki constraint
* Kadang, ada saatnya misal kita ingin melakukan pengecekan beberapa hal saja pada kondisi tertentu, misal terdapat satu class yang digunakan untuk beberapa aksi,sehingga membutuhkan kombinasi validasi yang berbeda
* Dalam kasus ini, kita bisa menggunakan Grouping Constraints
* Grouping Constraint merupakan teknik mengelompokkan constraint yang terdapat di class
* Degan menambahkan group, saat melakukan validasi, kitab isa memilih group mana yang akan divalidasi
* Setiap constraint bisa memiliki lebih dari saru group

Kode : Membuat Group dengan create class interface

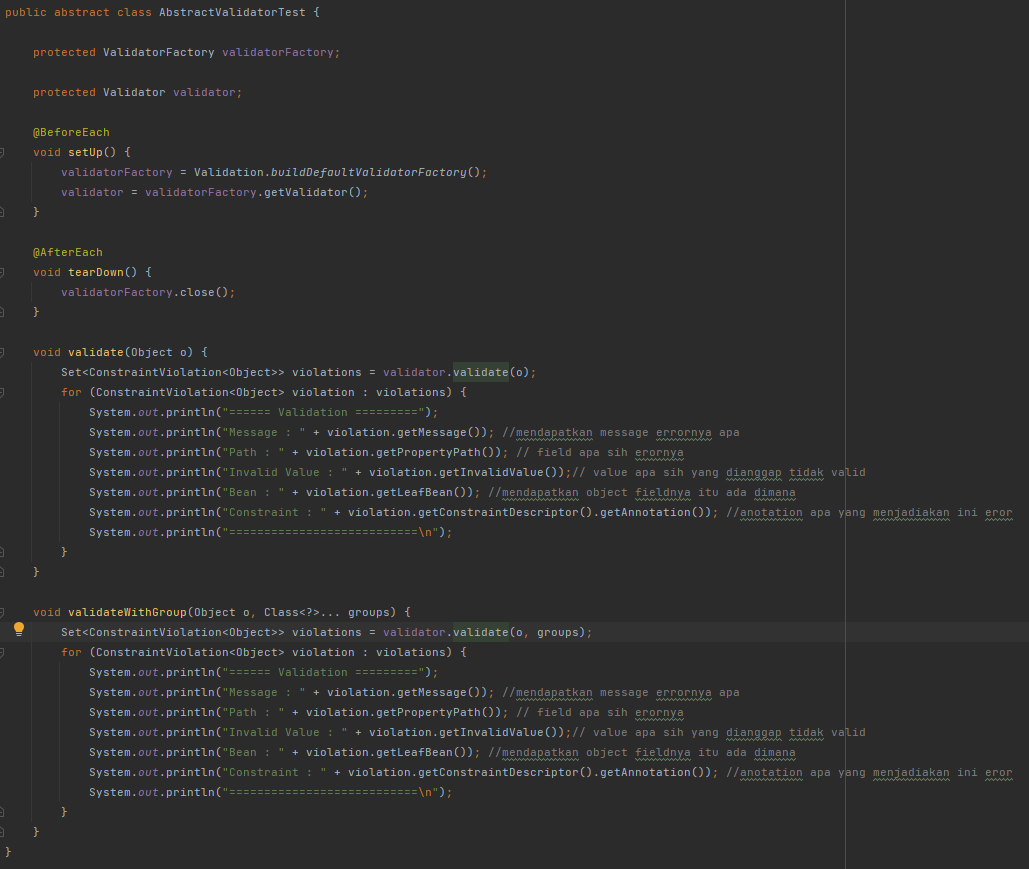




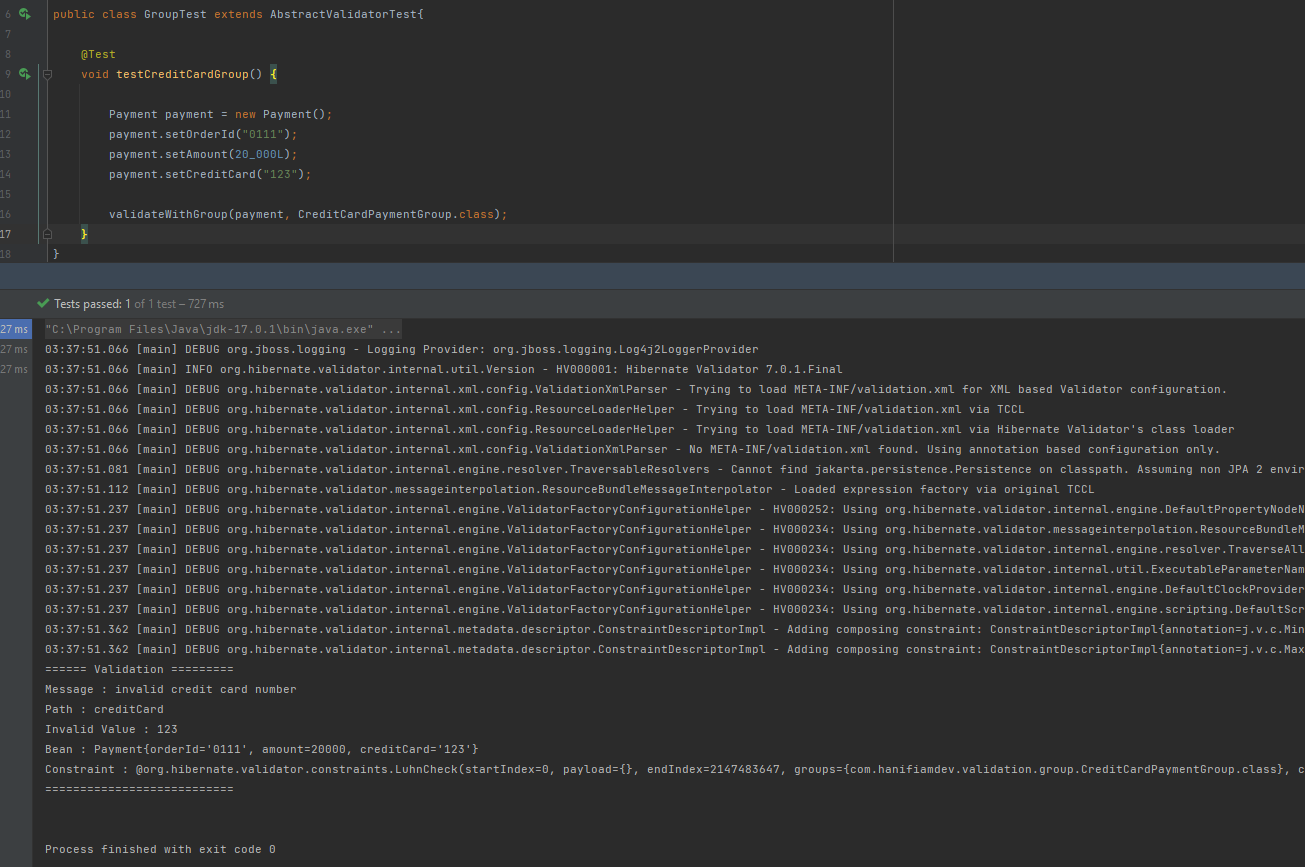
Add group di dalam bean validation, menambahkan field cirtualAccount beserta getter and setter



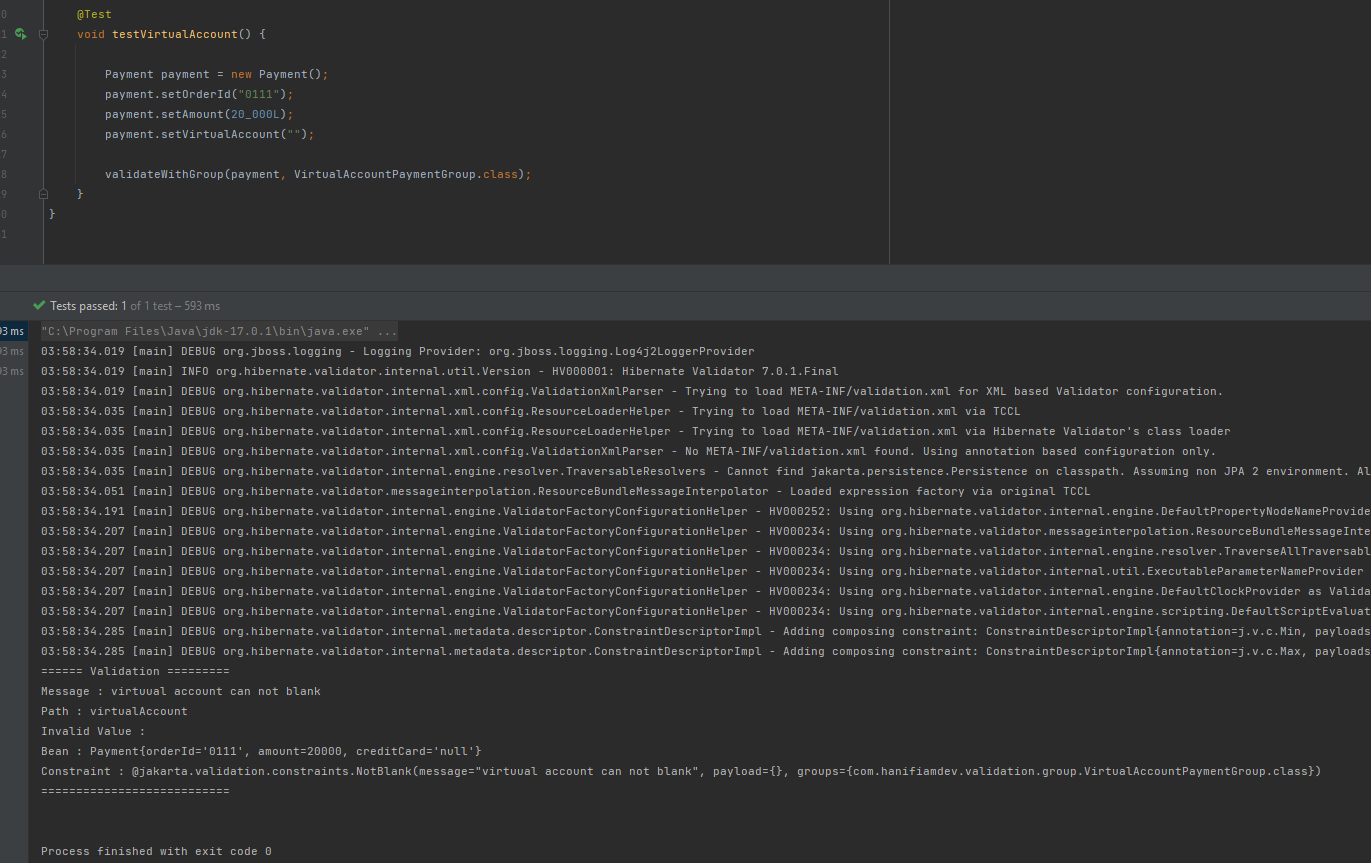
Sesuaikan kelas AbstractValidatorTest supaya bisa eksekusi dengan input param group



Test Validasi terhadap group CreditCardPaymentGroup.class



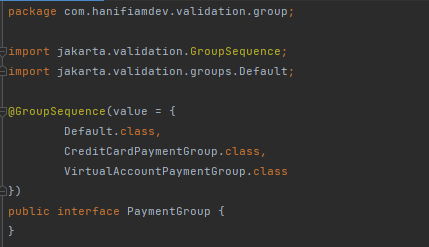
Test Validasi terhadap group CreditCardPaymentGroup.class

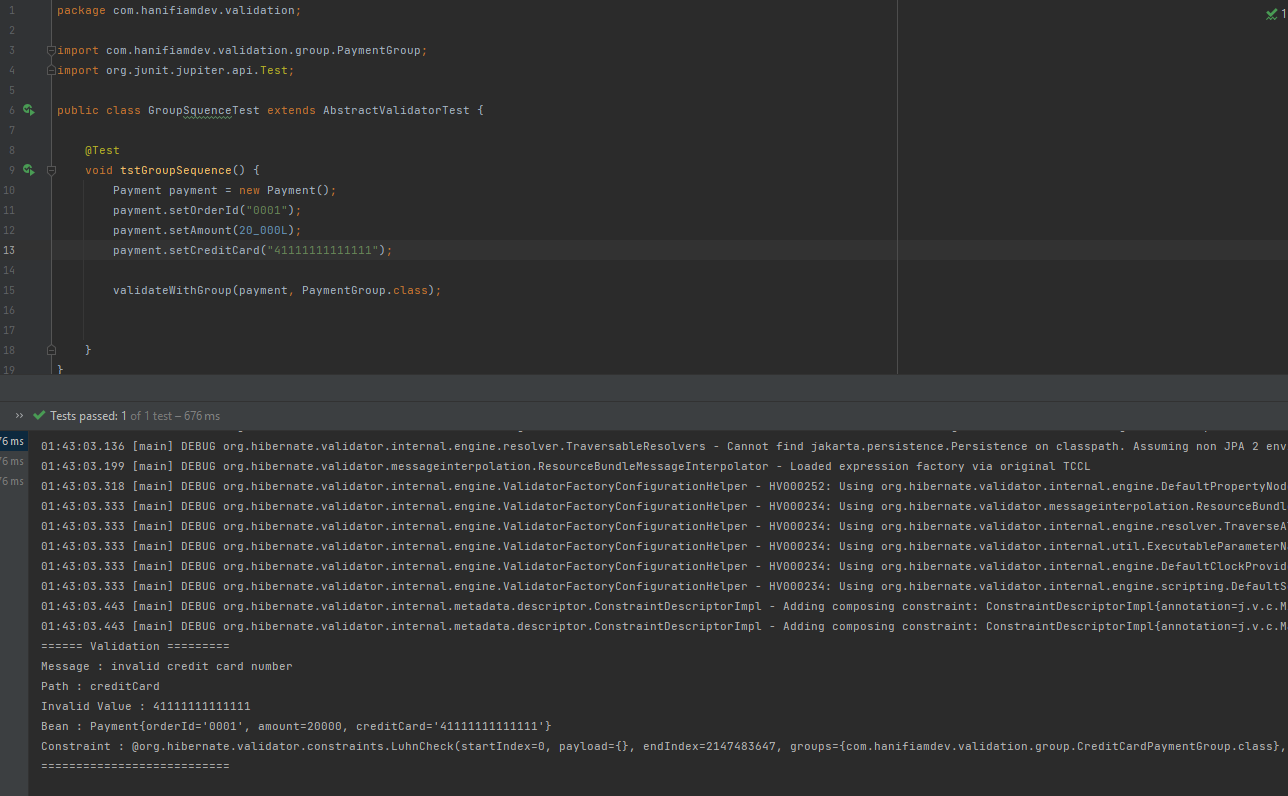


## **Group Sequence**

* Saat kita melakukan validasi dengan beberapa group, tidak ada jaminan bahwa sebuah group akan dijalankan sebelum group yang lain
* Bean Validation memiliki annotation GroupSequence, ini digunakan untuk menentukan tahapan group mana terlebih dahulu yang akan di validasi
* Kita bisa membuat group baru, lalu tambahkan annotation @GroupSequence, atau langsung di class nya
* Saat terjadi error validasi pada sebuah group, maka secara otomatis tidak akan dilanjutkan ke group selanjutnya

Mendaftarkan urutan validasi dengan membuat interface





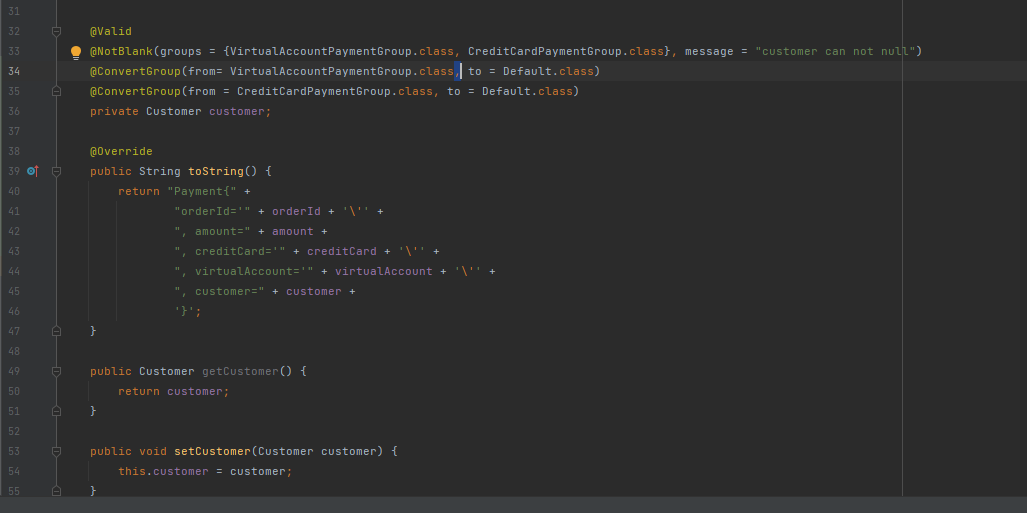
## **Group Conversion**

* Kadang ada kasus dimana terdapat sebuah class yang sudah memiliki field group, namun ternyata kita membutuhkan class tersebut di embed di class lain, sedangkan class lain menggunakan group berbeda
* Pada kasus seperti ini, kita bisa melakukan konversi group
* Untuk melakukan konversi group, kita bisa menggunakan annotation @ConvertGroup, lalu tentukan dari group apa ke group apa

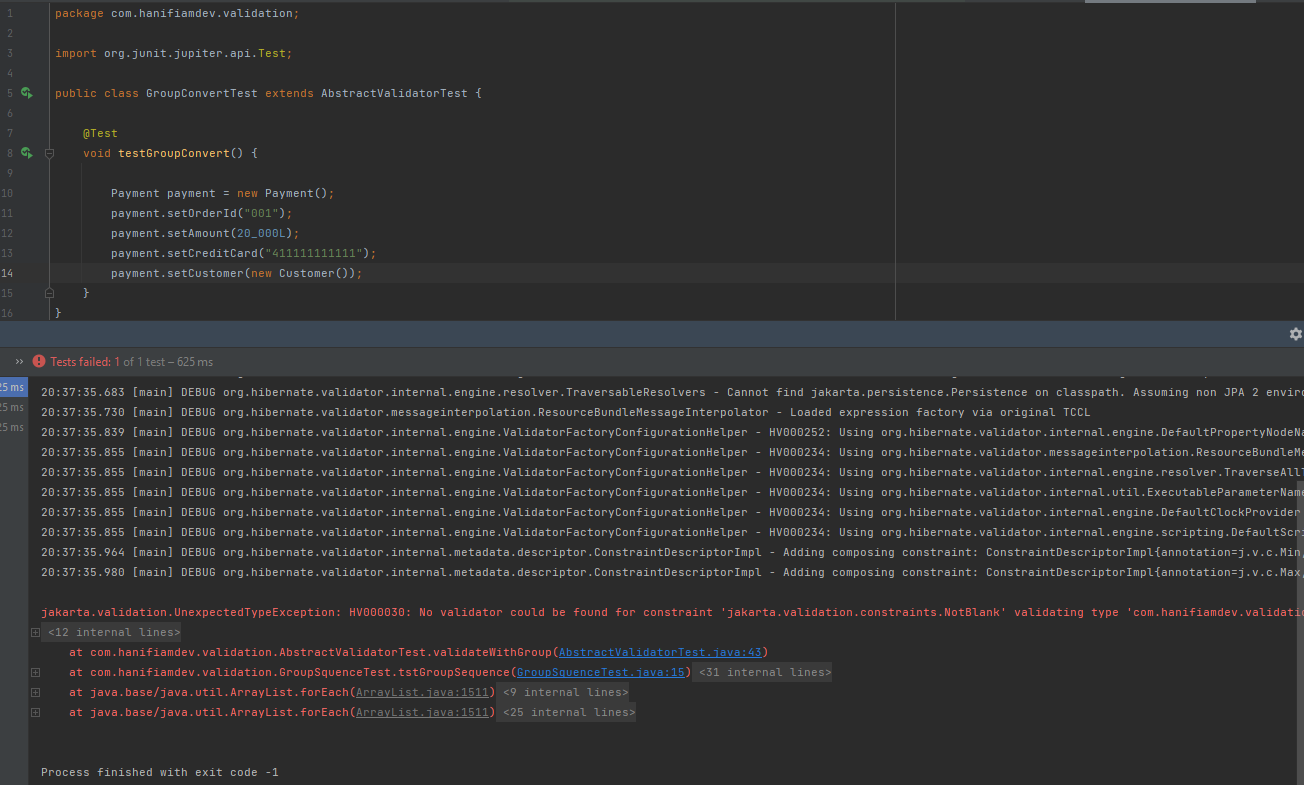
Buat kelas customer yang akan di sumulasi validasi dengan convert group



Tambahkan field beserta getter and setter dan ubah pula toString() nya untuk validasi dari kelas Payment. Karena kelas Customer validasinya Default maka perlu di convert ketika melakukan validasi dari group lain



Perhatikan walaupun dengan group selain Default masih tetap ikut divalidasi di kelas Customer



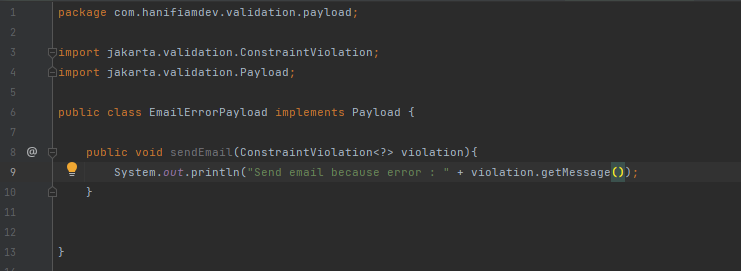
## **Payload**

* Secara default aturan Annotation Constraint di Bean Validation selain memiliki method message() dan groups(), ada juga payload()
* Untuk apa payload()?
* Method payload() itu sendiri sebenarnya tidak digunakan sama sekali oleh Bean Validation, namun method ini bisa digunakan oleh kita untuk menambahkan informasi ketika menggunakan constraint
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/payload>

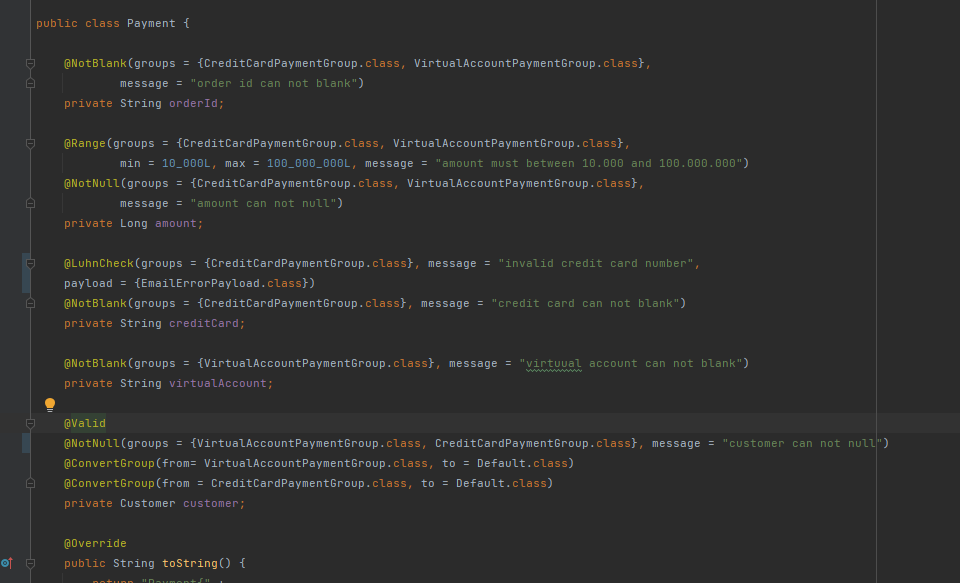
Membuat Payload

* Sekarang kita akan coba membuat sebuah payload yang sederhana
* Payload yang akan kita buat adalah sebuah class dimana jika terjadi validation error, kita ingin gunakan payload ini untuk mengirimkan misal email untuk membertahu jika terjadi error

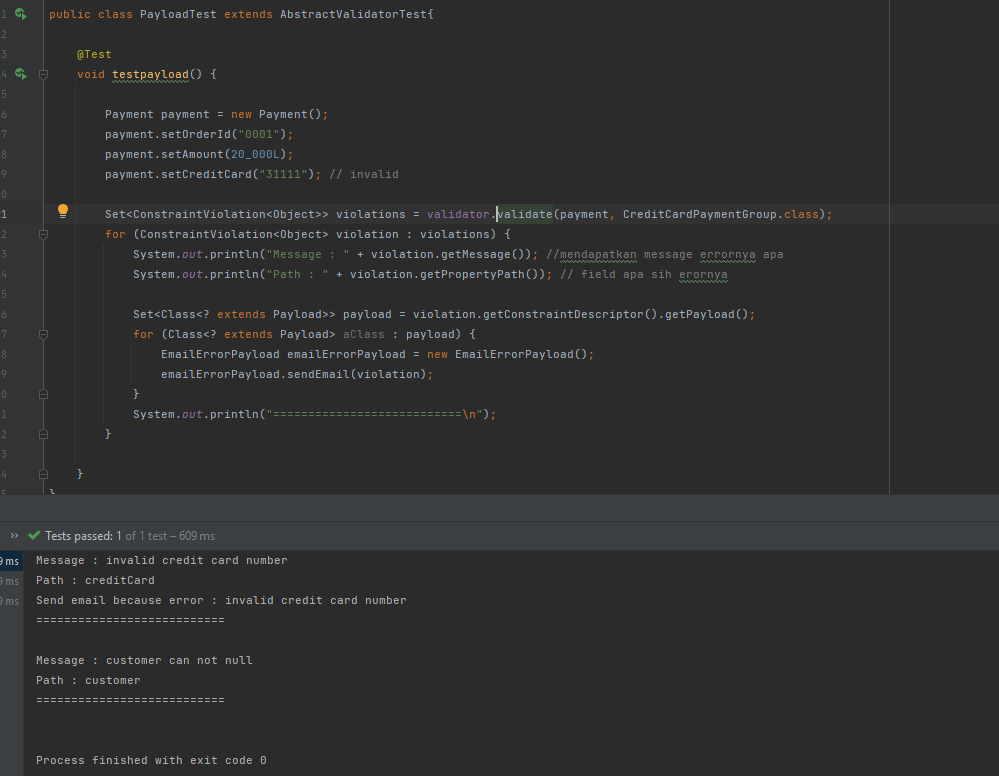
Kode : Interface EmailErrorPayload yang implement Payload



Tambahkan payload di validasi field creditcard kelas Payment



Test validasi dan Get Payload



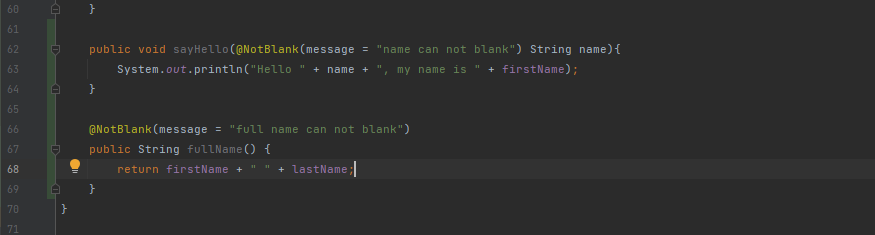
## **Method Validation**

* Sampai saat ini kita baru hanya melakukan validasi pada object yang kita buat secara manual
* Bean Validation sendiri bisa digunakan untuk melakukan validasi di Method, baik itu Method Parameter atau Return Value
* Fitur ini memudahkan kita karena cukup menambahkan Annotation Constraint di Method Parameter

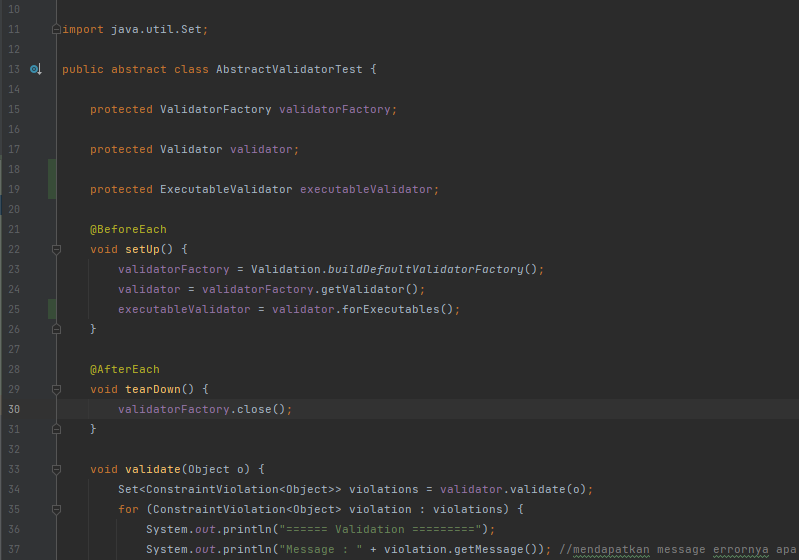
**ExecutableValidator**

* Untuk melakukan validasi di Method, kita butuh object ExecutableValidator
* Untuk membuat ExecutableValidator, kita bisa gunakan method for Executables() di Validator
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/executable/executablevalidator>

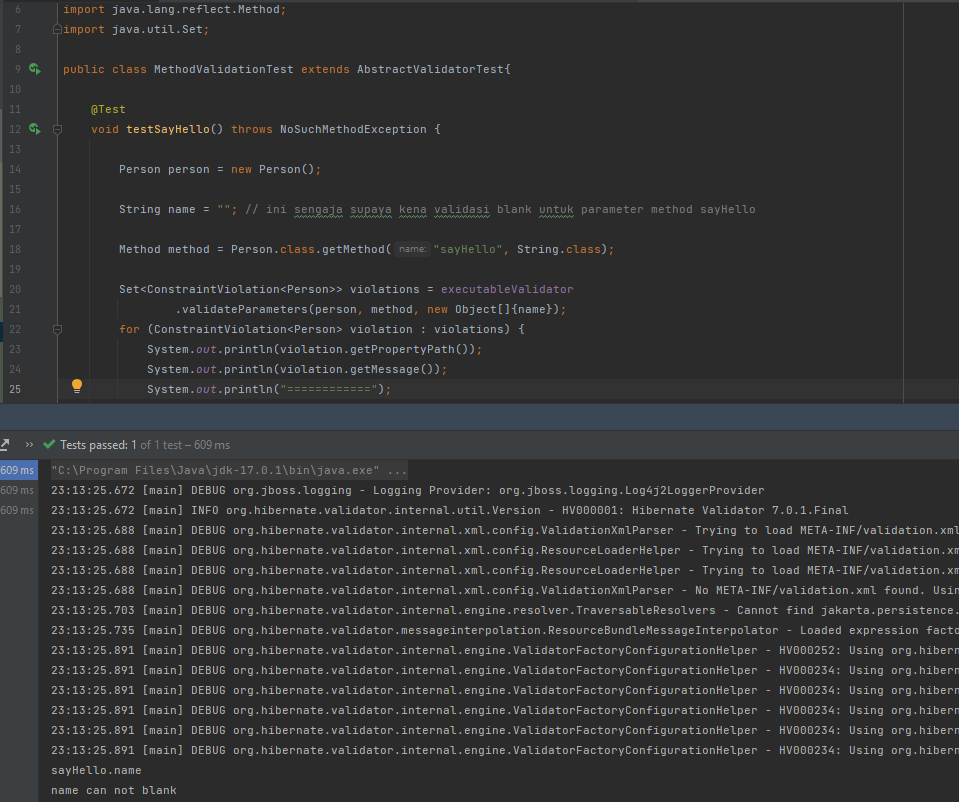
Add method sayHello ke Person class



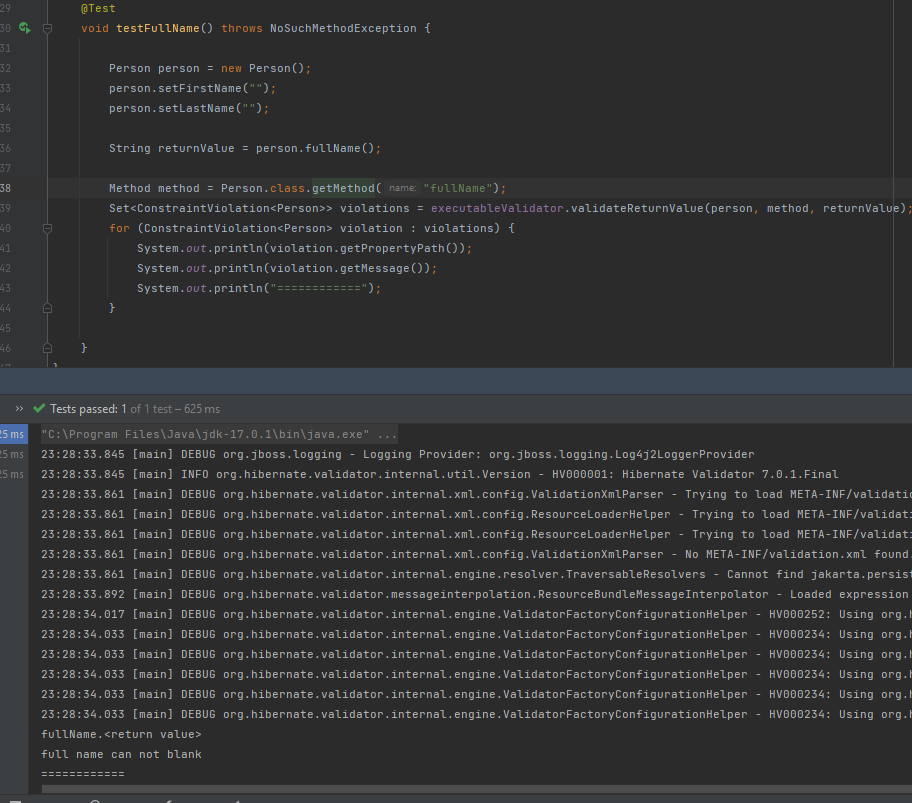
Add Executablevalidator untuk inisialisasi method validation



Test validasi parameter method



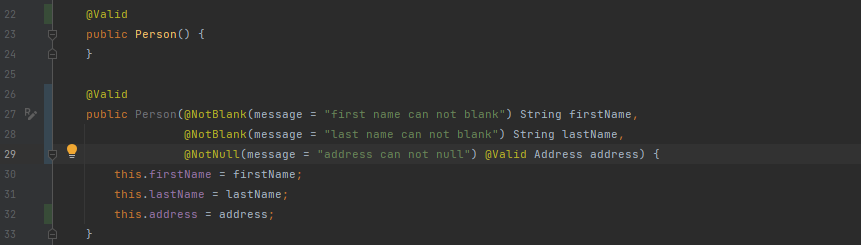
Test Kode : Validate Method Return Value



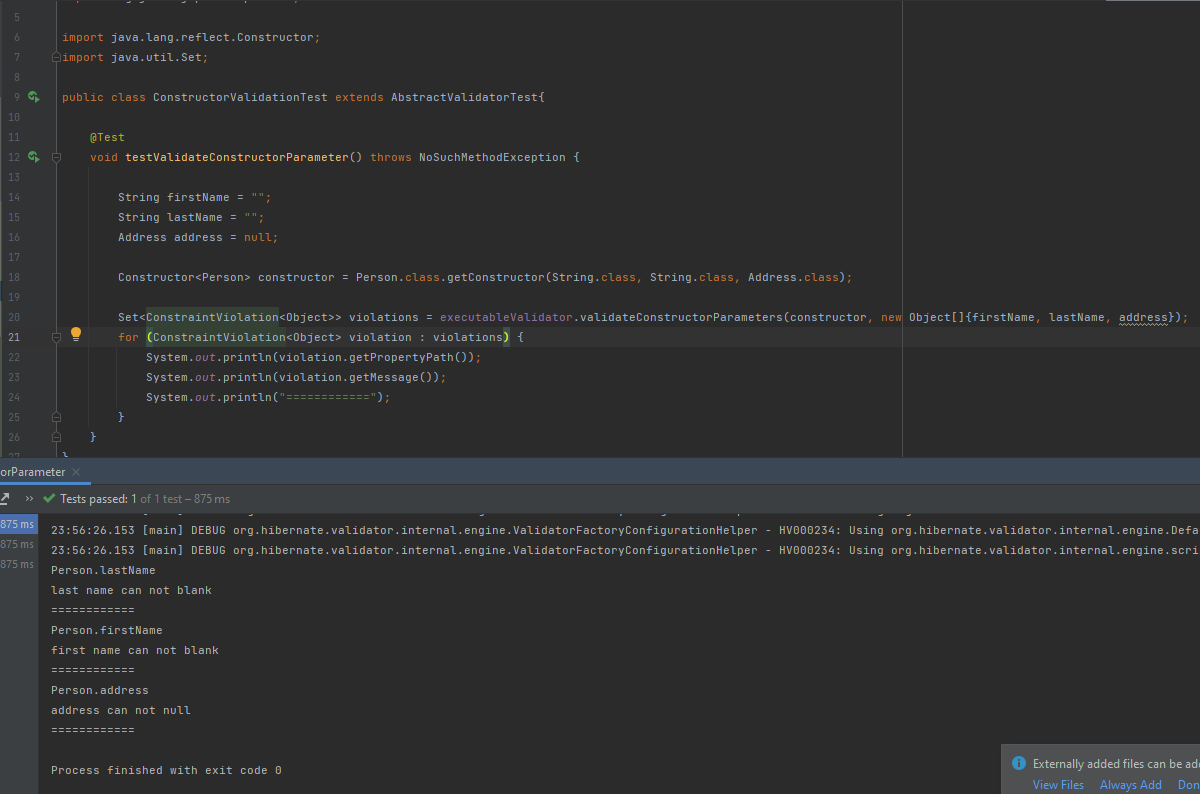
## **Method Validation**

* ExecutableValidator tidak hanya bisa digunakan untuk melakukan validasi terhadap Method Parameter dan Method Return Value, tapi juga ke Constructor Parameter dan Constructor Return Value
* Cara penggunaannya sama dengan melakukan validasi di Method, yang berbeda hanya method yang digunakannya

Modifikasi constructor kelas Person



Test Validasi Contructor parameter



Test Validasi Return Contructor



## **Message Interpolation**

* Message Interpolation merupakan proses membuat proses eror ketika terjadi kesalahan pada constraint
* Secara default, pesan kesalahan akan diambil dari method message() milik constraint

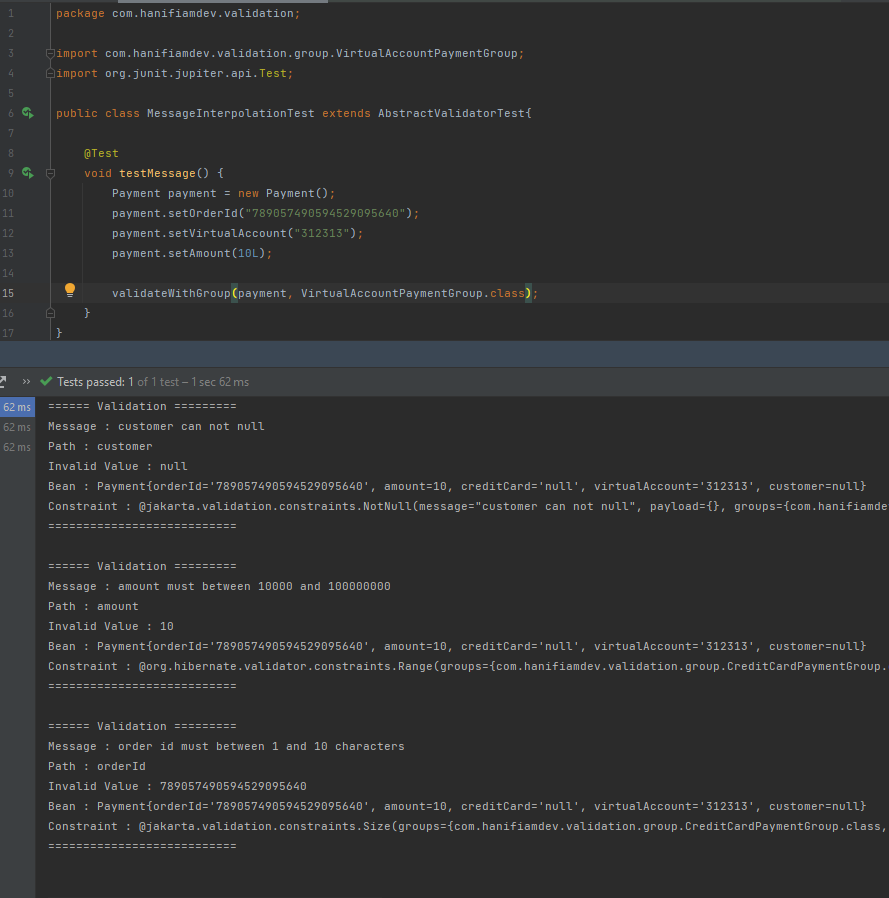
**Special Character**

* Message Interpolation memiliki karakter spesial yaitu { dan }, oleh karena itu jika kita ingin menggunakan karakter tersebut, kita perlu tambahkan \ didepannya, misal \{ atau \}
* Kadang ketika kita membuat pesan kesalahan , kita ingin mengambil value dari constraint nya, kita bisa menambahkan {method} yang terdapat di constraint nya, secara otomatis nilai di constraint akan ditambahkan ke message nya

Add contoh message interpolation untuk validasi @SIZE di orderId dan amount



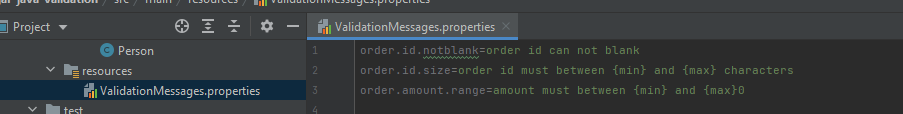
Test



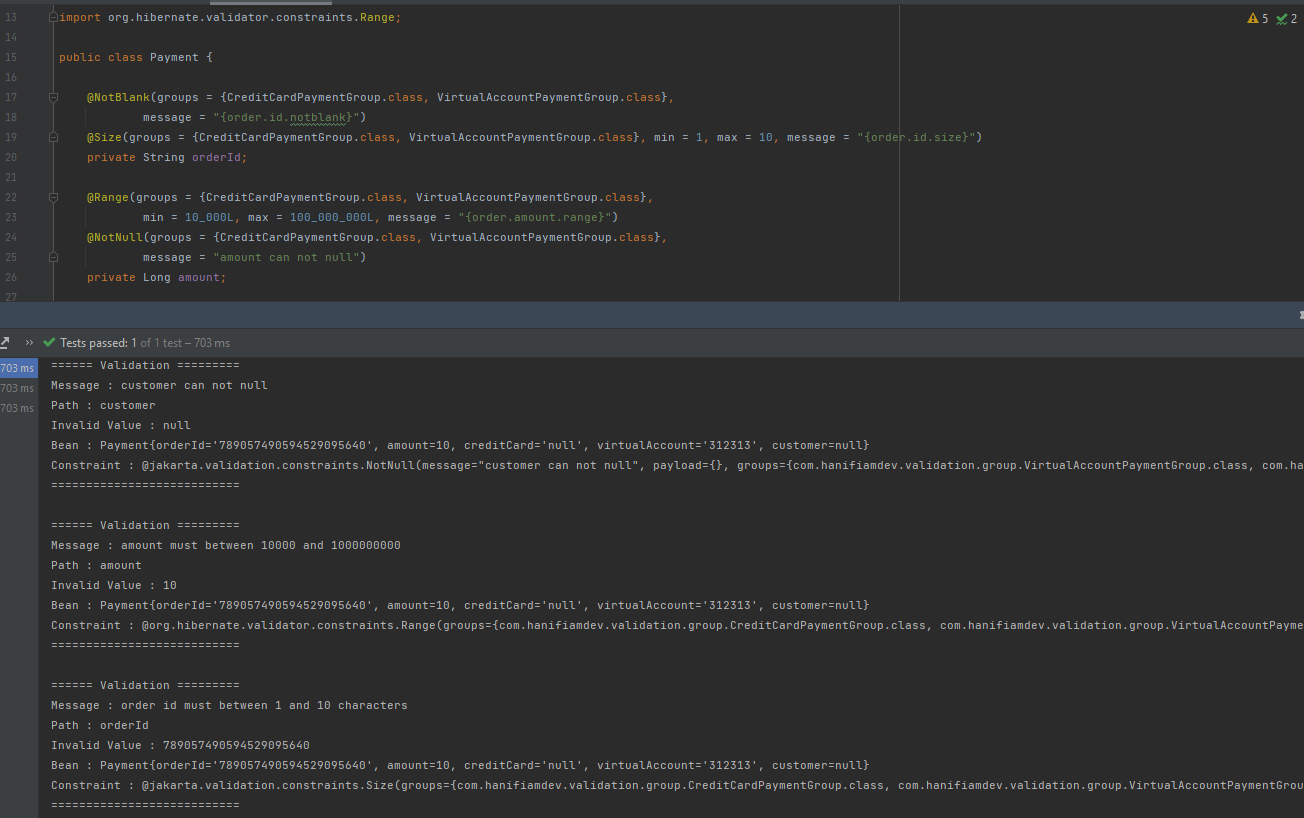
**Resource Bundle**

* Selain hardcode pesan di dalam method message(), Bean Validation juga mendukung resource bundle, dimana kita bisa menyimpan semua pesan kesalahan di file properties
* Hal ini sangat bagus ketika kita butuh mendukung pesan kesalahan dengan beberapa bahasa
* Caranya kita cukup buat file ValidationMessage.properties

Buat ValidationMessage.properties ( nama file wajib seperti ini ) di dalam folder resources



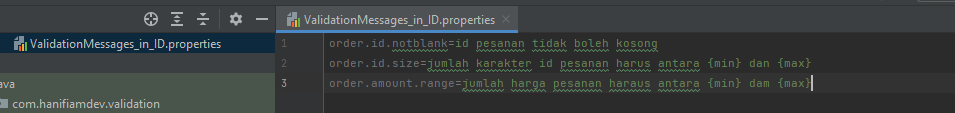
Test

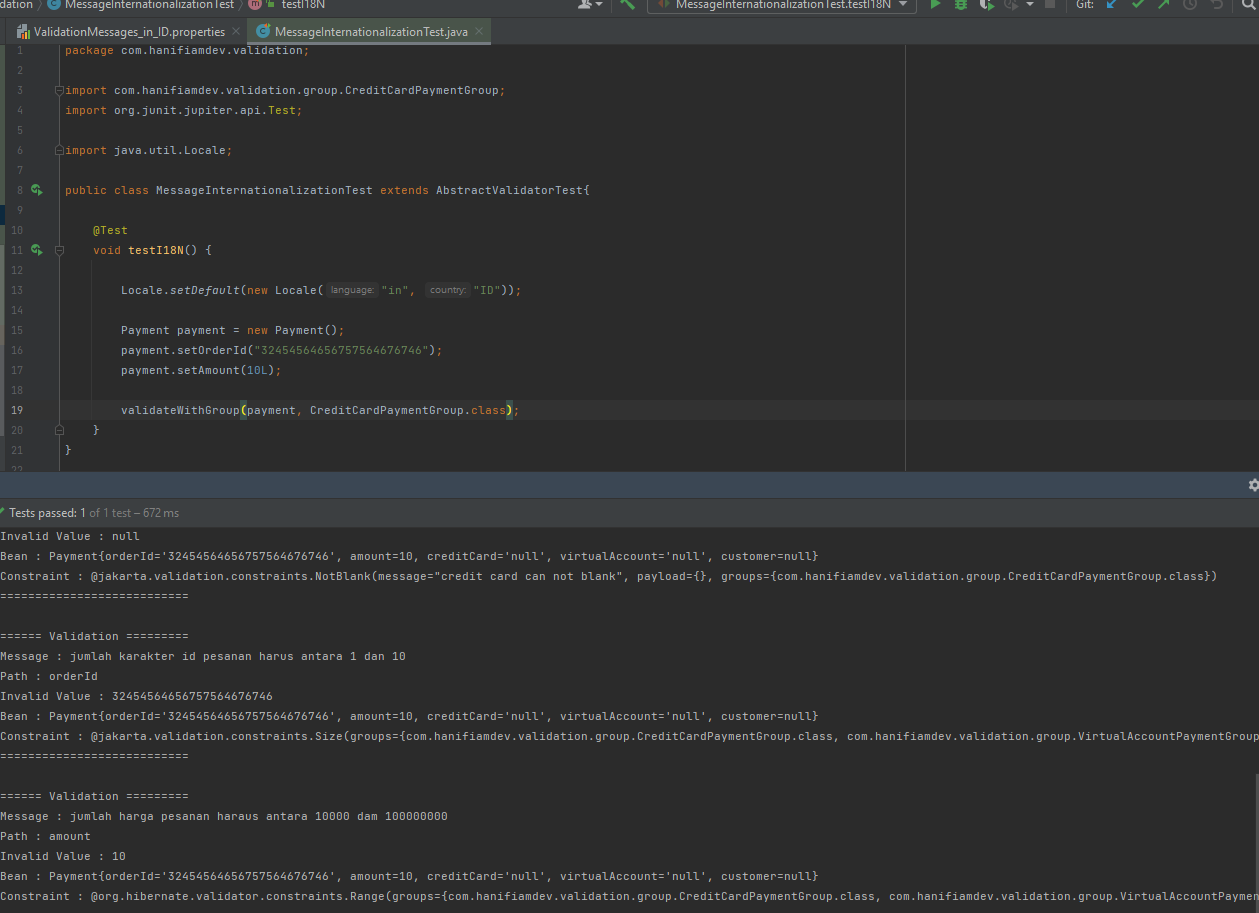


## **Message Internationalization**

* Secara default saat kita menggunakan Resource Bundle, Localle yang akan dipilih adalah Locale.getDefault()
* Jadi jika kita ingin mengubah Locale untuk Resource Bundle,kita harus mengubah default Locale nya

Buat ValidationMessage\_in\_ID.properties ( nama file wajib seperti ini ) di dalam folder resources

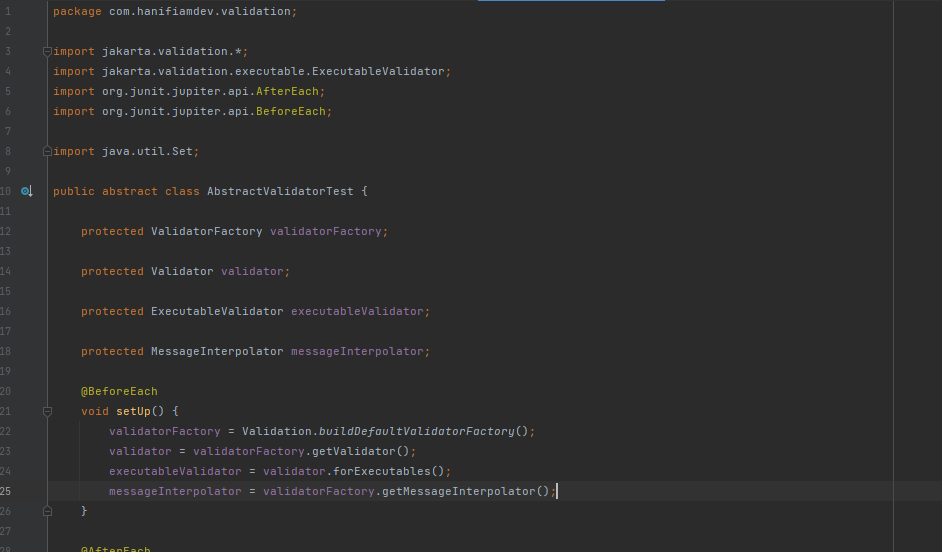




## **MessageInterpolator**

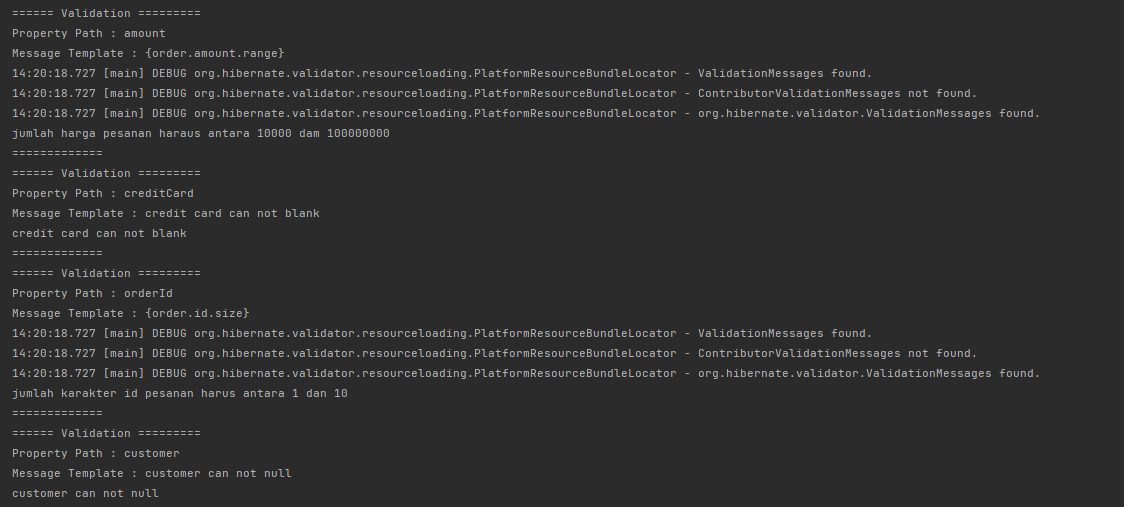
* Salah satu cara lumayan sulit, namu lebih flexible adalah menggunakan MessageInterpolator secara langsung
* Cara ini sangat flexible, tapi sangat sulit karena kita harus membuat context secara manual untuk MessageInterpolator
* Kita tidak perlu mengubah Default Locale, hanya cukup gunakan parameter Locale di Message Interpolator
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/messageinterpolator>

Add message interpolator





Result



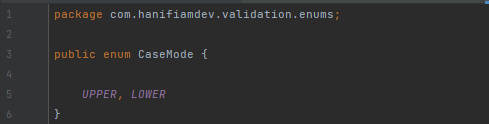
## **Custom Constraint**

* Sampai saat ini kita hanya menggunakan Constraint Built-In yang terdapat di Bean Validation dan Hibernate Validator
* Bagaimana jika ternyata Constraint Built-In tersebut tidak bisa memenuhi kebutuhan validasi kita?
* Maka kita bisa membuat Constraint sendiri, atau istilahnya Custom Constraint
* Untuk membuat Constraint, kita perlu mengikuti aturan nya, pertama kita buat Constraint Annotation, lalu kita buat ConstraintValidator

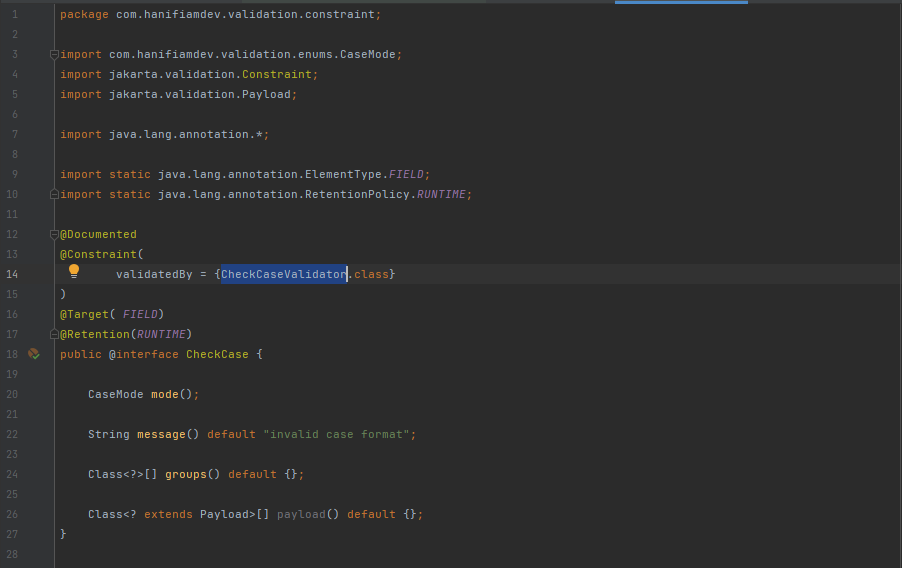
**Check Case Constraint**

* Misal, sekarang kita akan membuat sebuah constraint validation yang digunakan untuk melakukan pengecekan case sebuah string
* Agar dinamis, kita akan tambahkan method mode() nya di constraint, yang bisa dipilih apakah harus UPPER atau lower

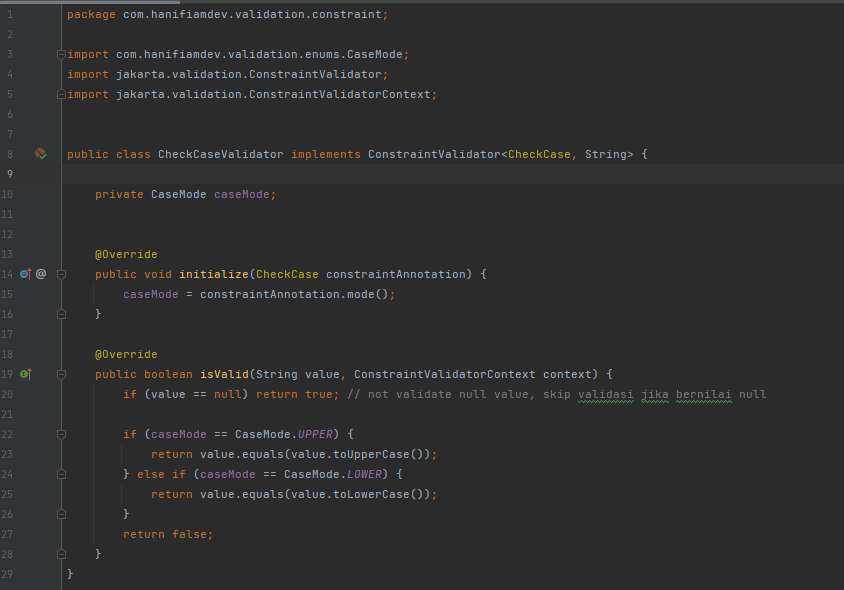
Kode : Enum CaseMode



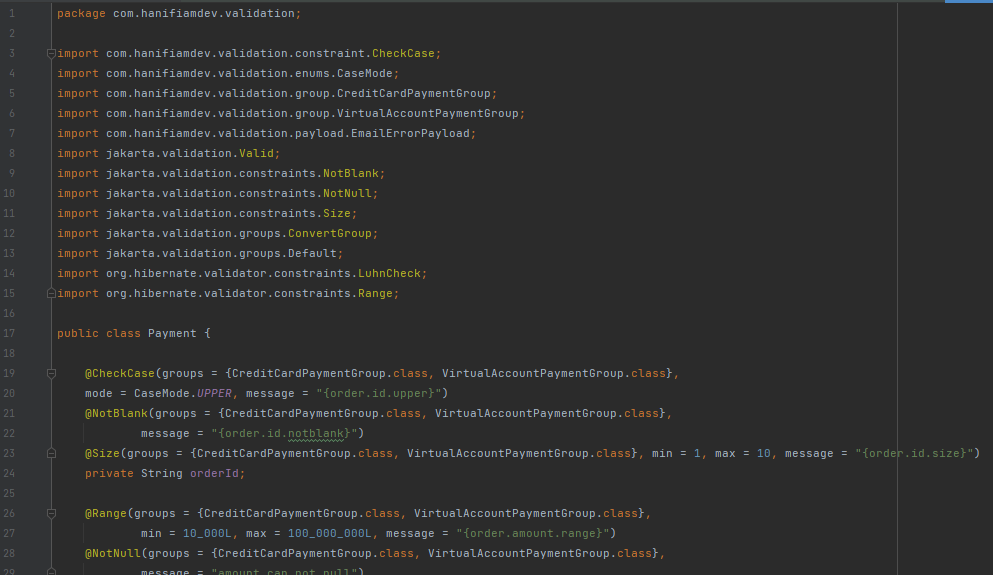
Kode : Annotation CheckCase



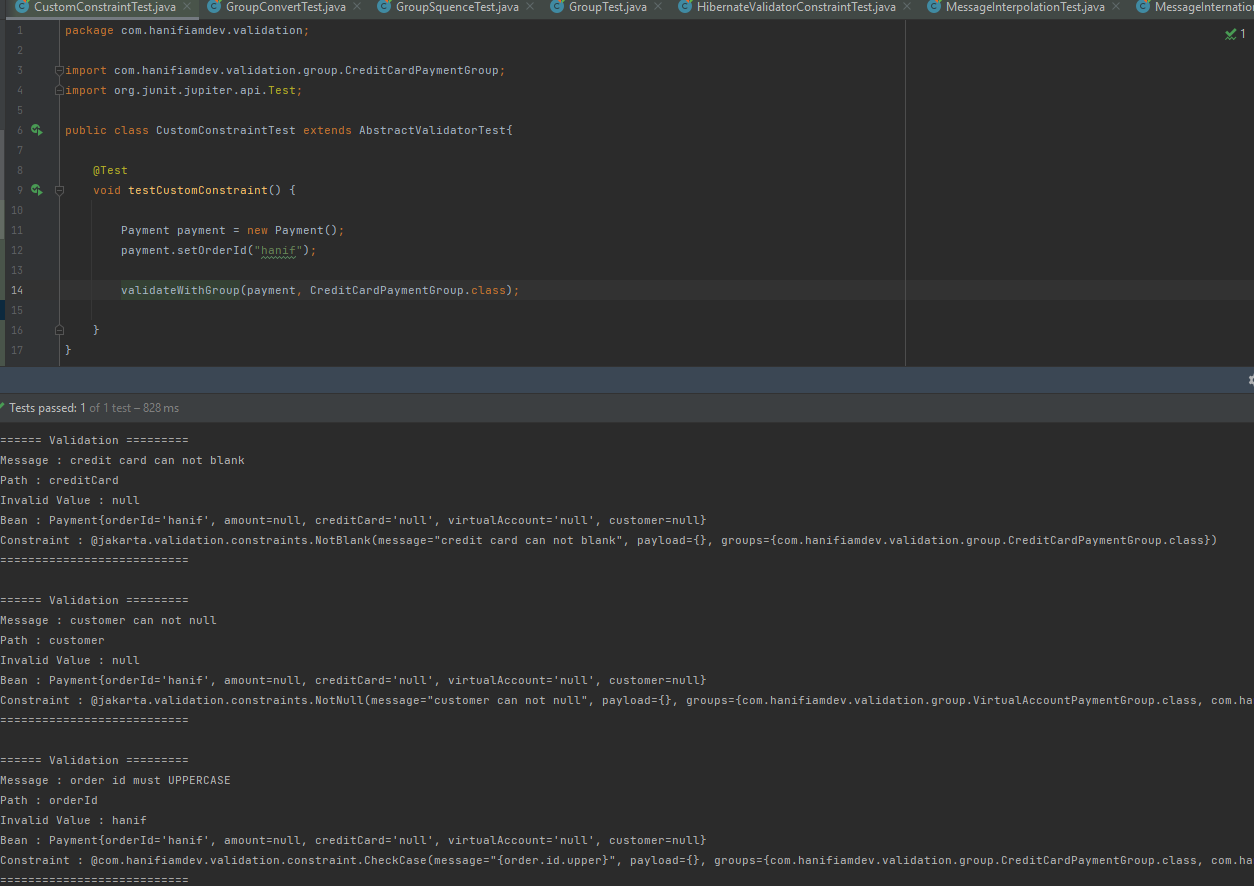
Kode : Class CheckCaseValidator (1)



Kode : Include validasi dengan @CheckCase di kelas Payment



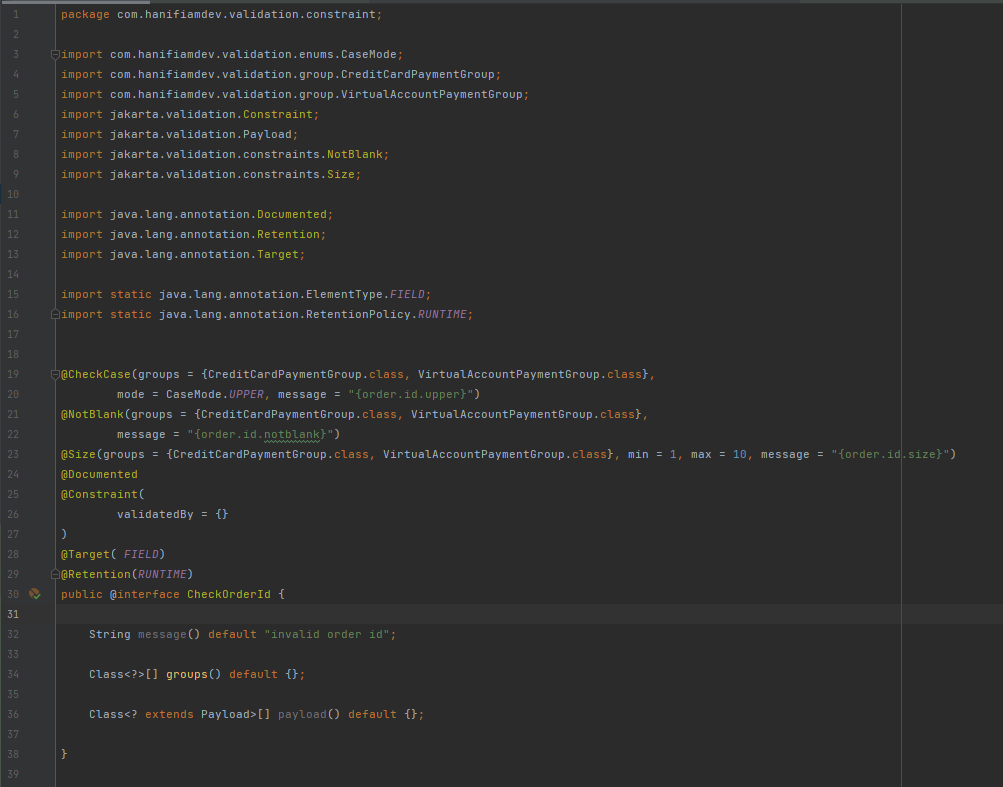
Kode : Unit Test



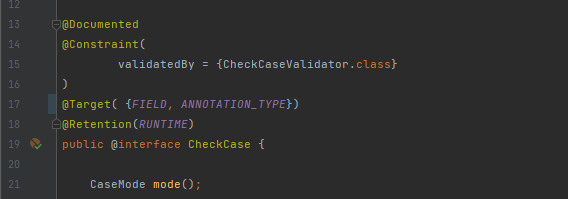
## **Constraint Composition**

* Jika kita perhatikan pada Order.id, kita lihat terdapat 3 constraint annotation, @Size, @NotBlank dan @CheckCase. Bahkan pada scenario tertentu, bisa jadi akan sangat banyak sekali constraint pada sebuah field, yang semakin lama akan semakin membingungkan
* Bean Validation mendukung Constraint Composition, dimana kita bisa membuat Constraint baru yang didalamnya sebenarnya adalah kumpulan Constraint lainnya
* Caranya sangat mudah, kita cukup membuat Constraint Annotation, lalu tambahkan Constraint Annotation lain pada Constraint yang kita buat

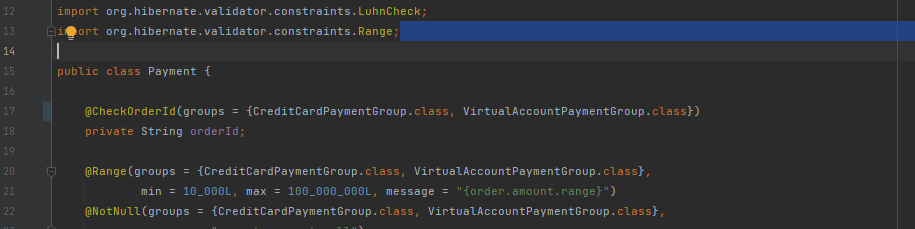
Kode : Constraint CheckOrderId



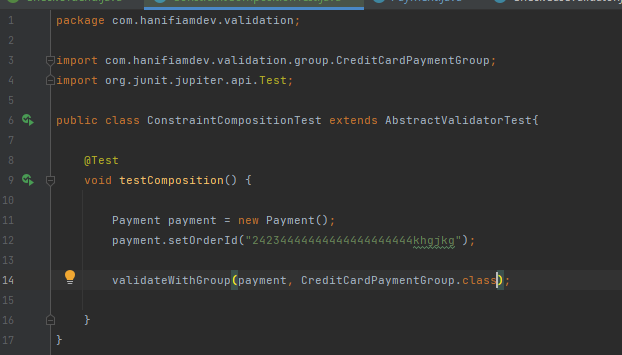
Add ANNOTATION\_TYPE ke dalam target



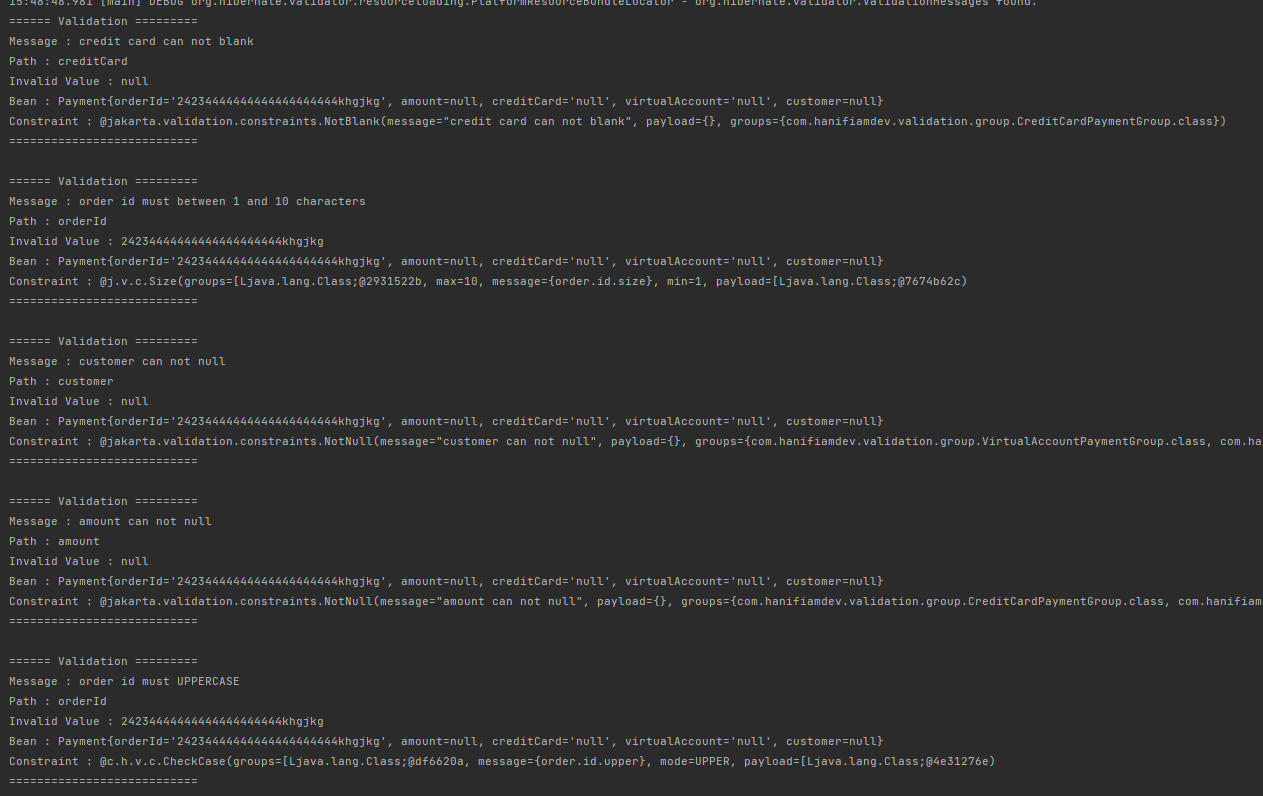
Add Constraint CheckOrderId ke dalam field orderId



Kode : Unit test



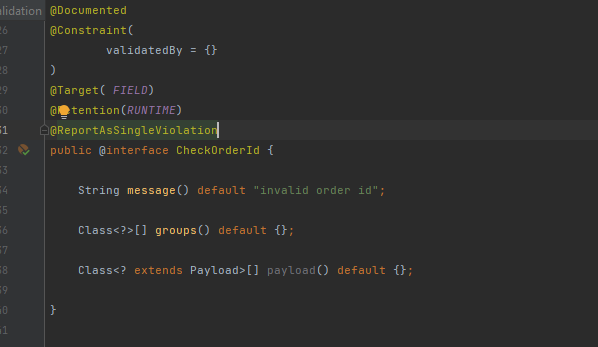
Result



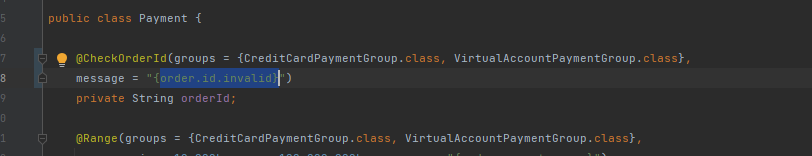
**@ReportAsSingleViolation**

* Secara default, semua Constraint di Constraint Composite akan dieksekusi, jadi misal terdapat lebih dari satu error, maka semua error akan diberitahukan
* Kadang ada baiknya, jika misal terdapat satu constraint yang error, kita tidak perlu lakukan pengecekan ke constraint berikutnya
* Untuk melakukan hal ini, kita bisa tambahkan annotation @ReportAsSingleViolation pada Composite Constraint Annotation

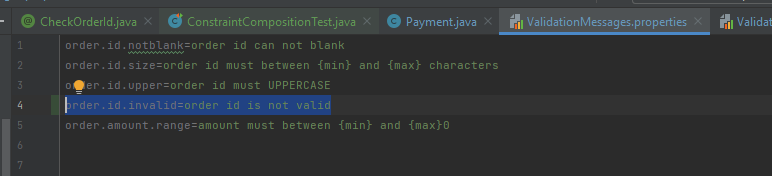
Add @ReportAsSingleCiolation

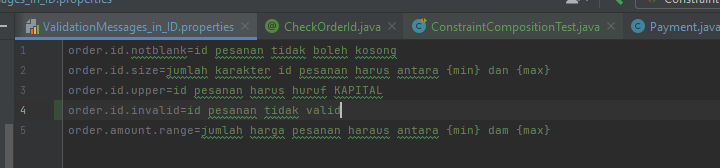


Add untuk single message orderId

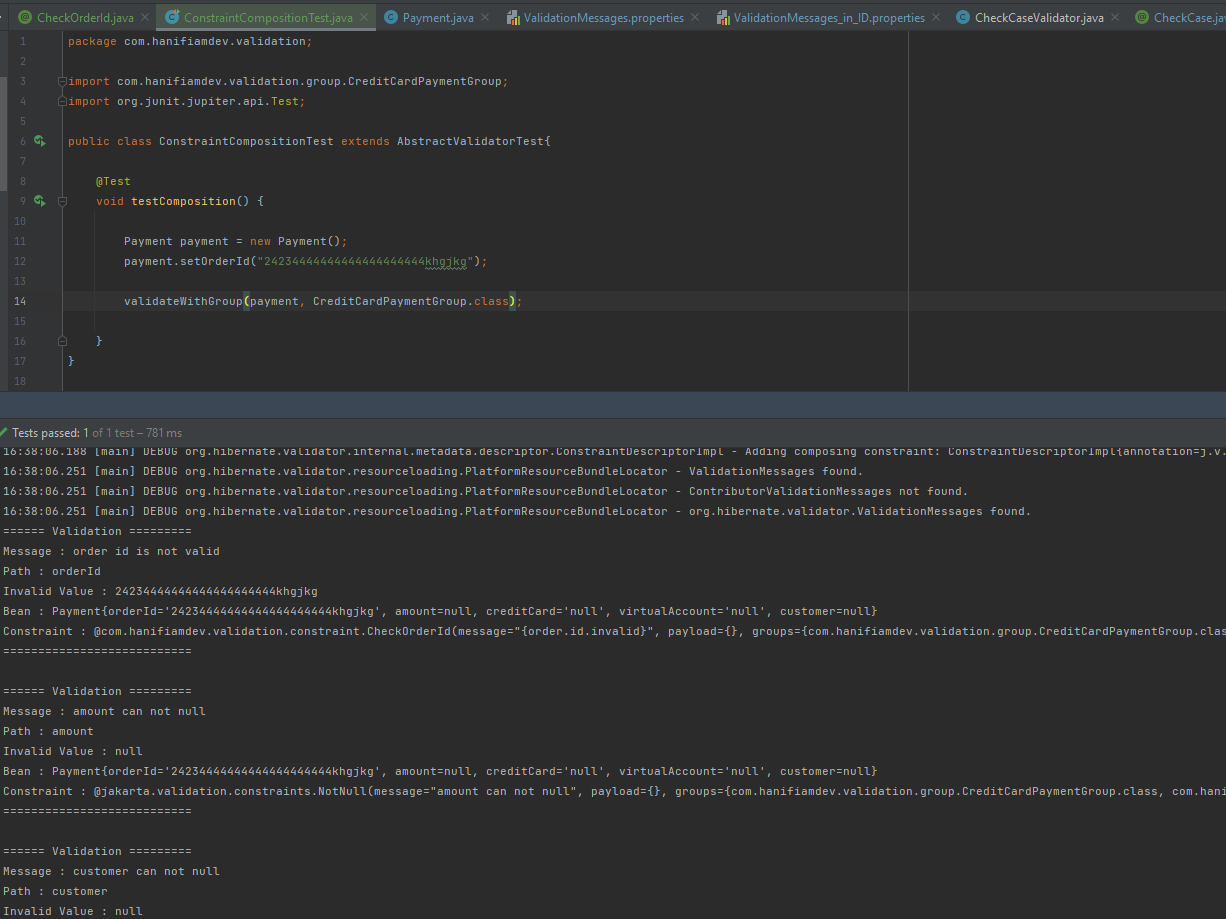


**Add detail message di properties**





Test, hanya terkena 1 kali saja kesalahan, karena validasi berikutnya di order id akan di ignore



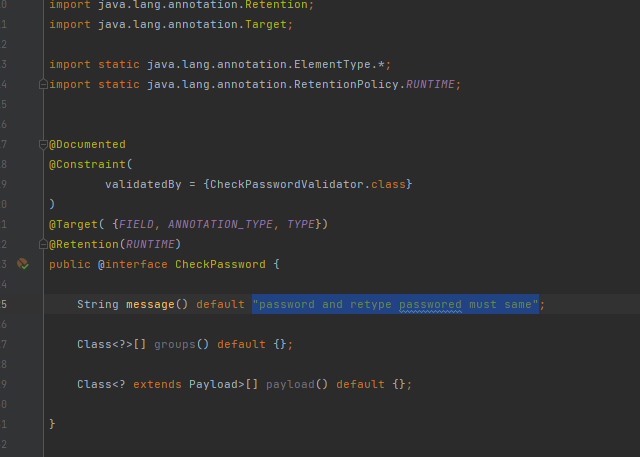
## **Class Level Constraint**

* Sebelumnya kita hanya membuat Constraint pada Field, Method, dan Constructor
* Constraint juga kita tambahkan pada level class
* Hal ini kadang bermanfaat, seperti misal kita ingin membandingkan lebih dari satu Field misalnya
* Untuk membuat Class-Level Constraint, kita cukup tambahkan Annotation Constraint pada Class, dan pastikan saat membuat Constraint, tambahkan terget Class

**Kode : Class Register**

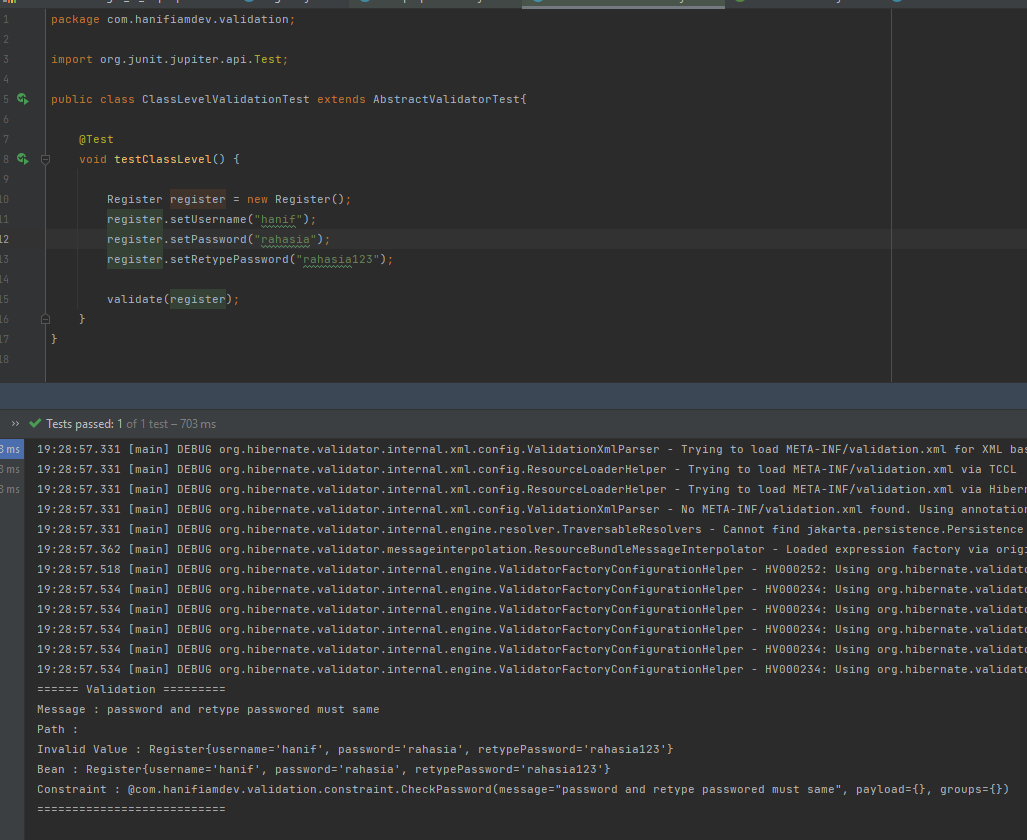


Kode : Constraint CheckPassword





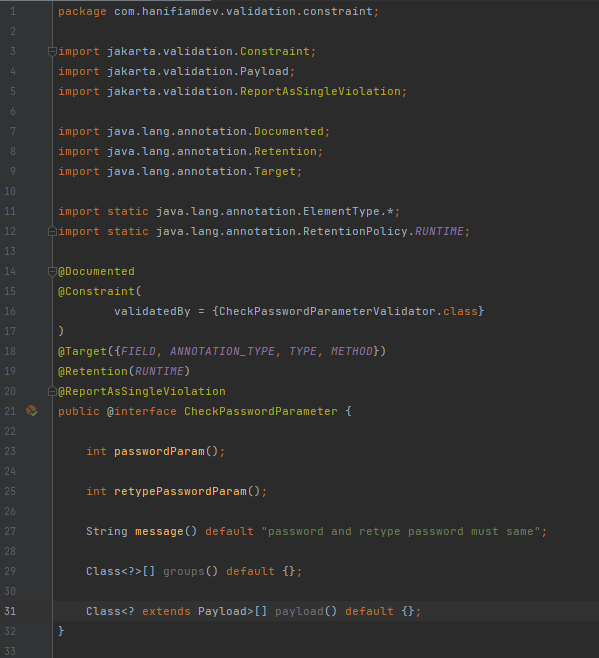
Test



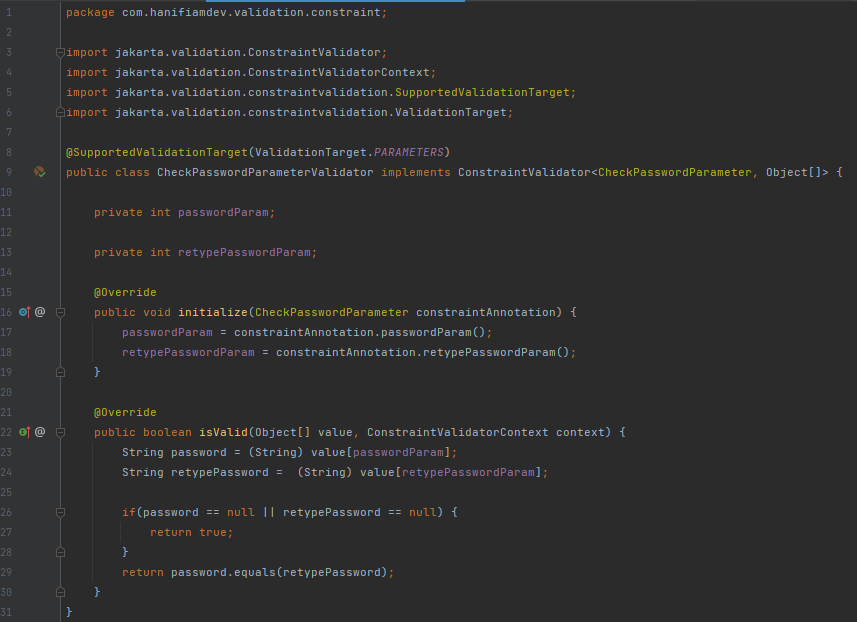
## **Cross-Parameter Constraint**

* Untuk melakukan validasi beberapa Field, kita bisa menggunakan fitur Class-Level Constraint
* Sekarang bagaimana jika kita ingin melakukan validasi beberapa Parameter? Misal pada Method Parameter atau Constructor Parameter
* Hal ini tidak bisa menggunakan Class-Level Constrain, namun ada cara sendiri untuk melakukan hal ini
* Yaitu dengan menggunakan Annotation @SupportedValidationTarget
* Kita bisa menggunakan @SupportedValidationTarget pada ConstraintValidator, untuk melakukan validasi parameter di Method atau Constructor

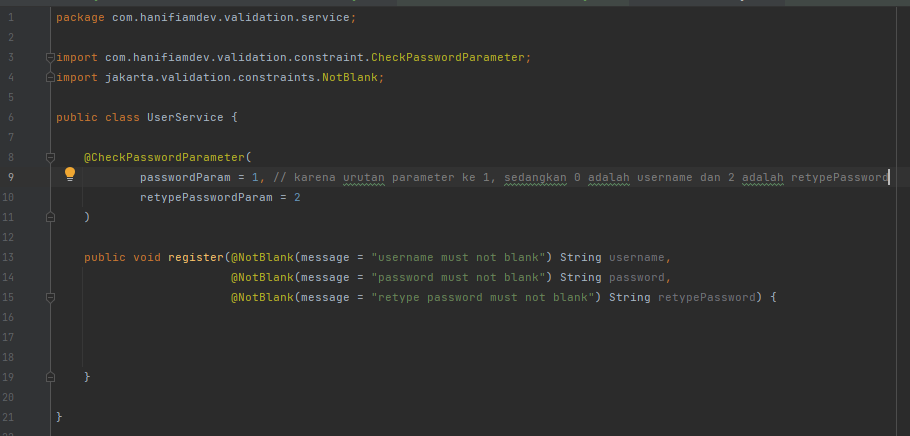
Kode : Check Password Parameter Constraint



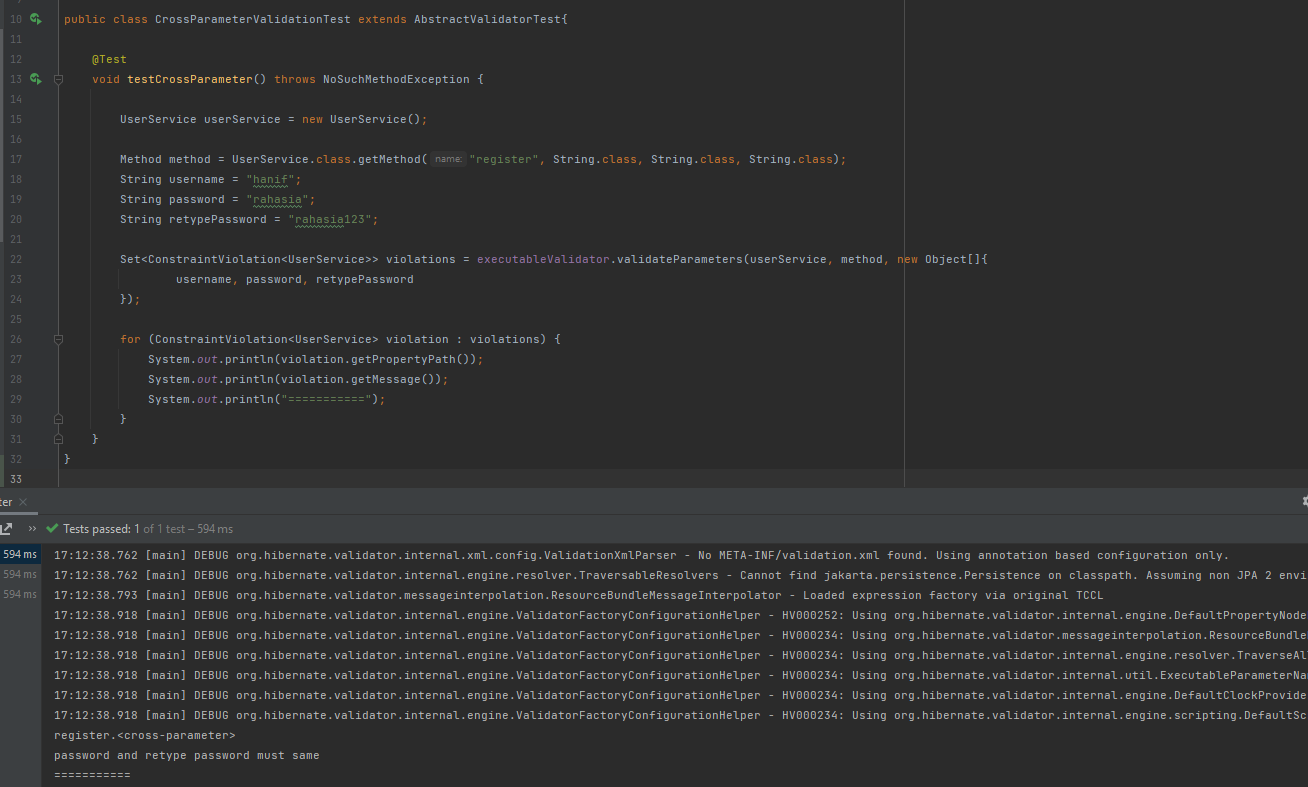
Kode : Check Password Parameter Validator



Kode : Register Method



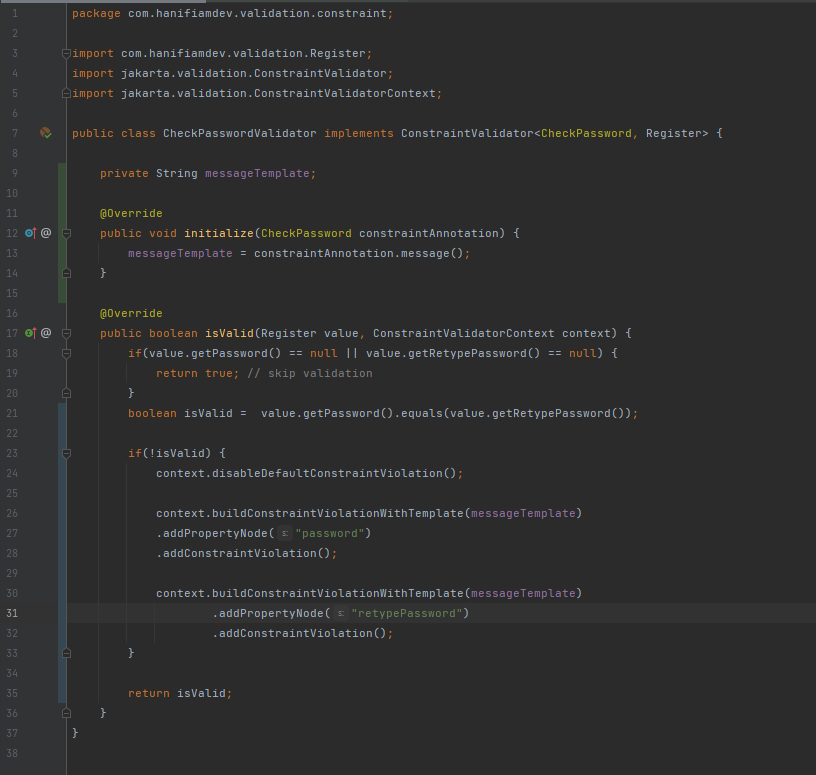
Kode : Cross-Parameter Constraint



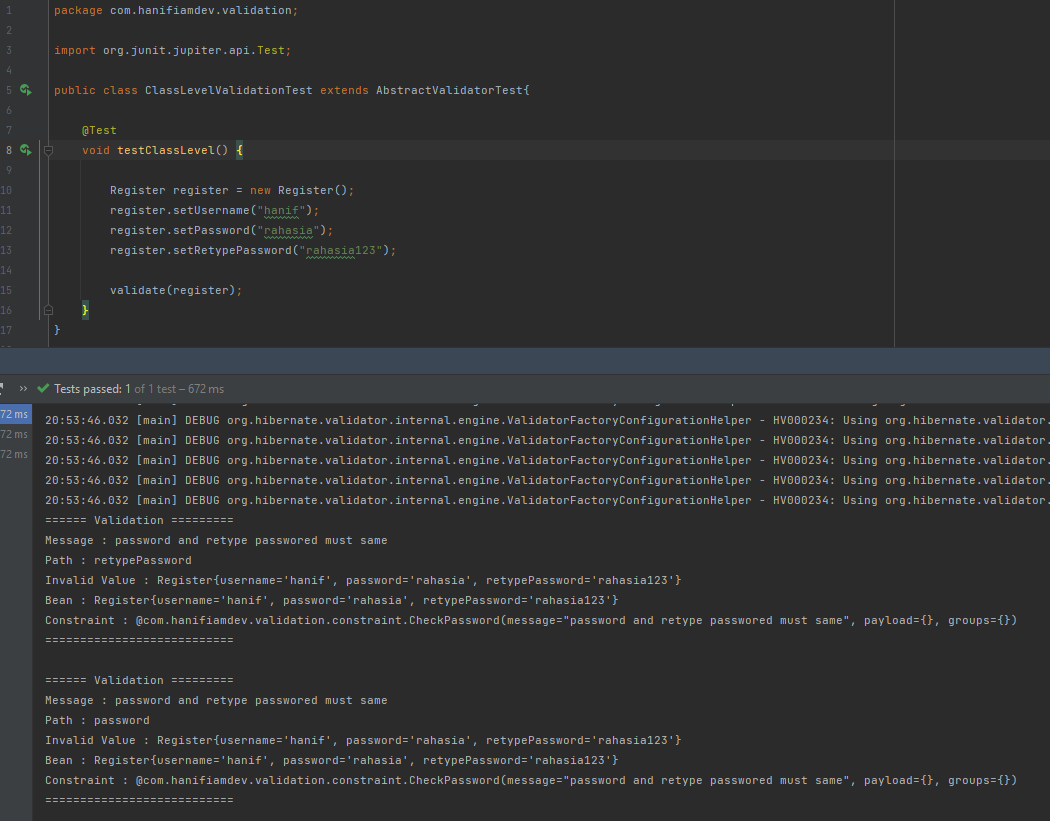
## **Constraint Validator Context**

* Saat kita membuat ConstraintValidator, pada method isValid terdapat parameter ConstraintValidatorContext
* Sebenarnya dalam keadaan ideal, kita tidak butuh menggunakan parameter tersebut, tapi pada materi ini kita akan bahas kegunaannya
* ConstraintValidatorContext digunakan untuk membuat custom message atau path, jika dirasa memang diperlukan
* Contoh pada kasus Class-Level Constraint, ketika terjadi error, maka secara default path nya adalah root objectnya, kadang-kadang kita ingin memberi tahu field mana yang error. Pada kasus ini kitab isa menggunakan ConstraintValidatorContext
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraintvalidatorcontext>

Kode : Constraint Validator Context, modify kelas CheckPasswordValidator.java



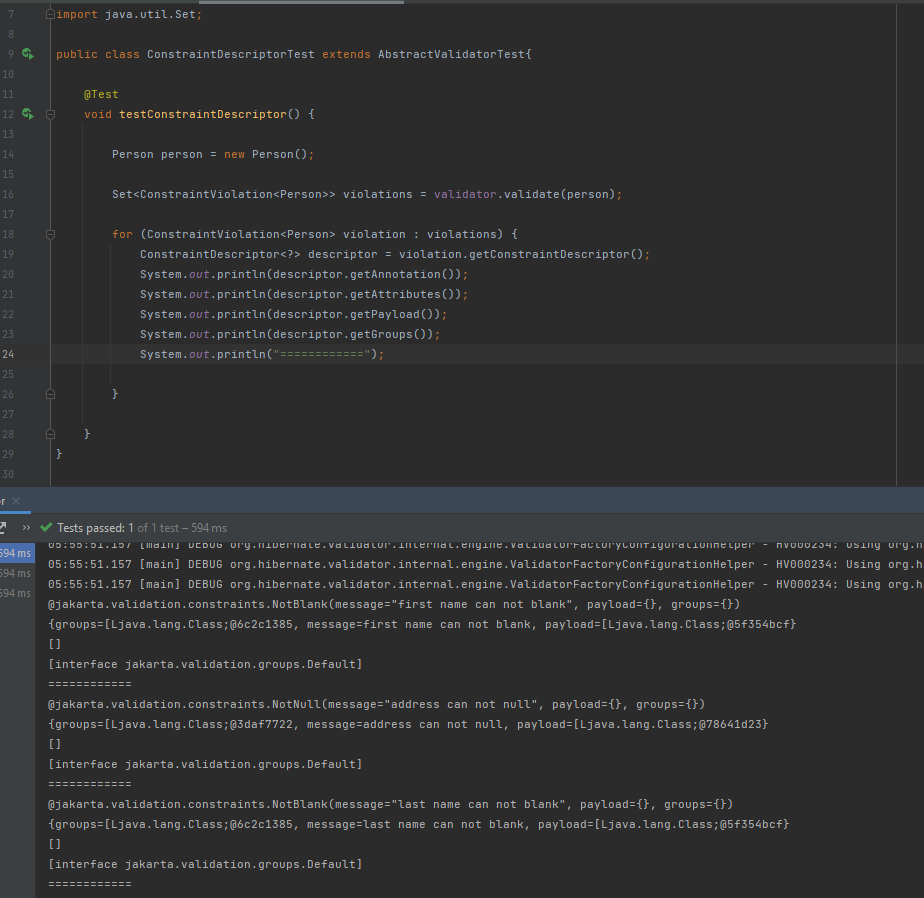
Test, maka sudah keluar property pathnya



## **Constraint Validator Context**

* Pada materi-materi sebelumnya, beberapa kali kita sudah menggunakan Constraint Descriptor, namun belum terlalu detail, seperti contohnya ketika mendapatkan Payload
* ConstraintDescriptor merupakan sebuah interface, yang berisikan informasi tentang Constraint
* Kita bisa mendapatkan informasi seperti Annotation, Group, Validator Class, Message Template, bahkan Annotation Method menggunakan ConstraintDescriptor
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/metadata/constraintdescriptor>

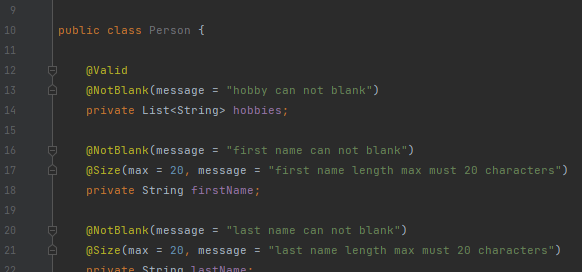
Kode : Constraint Descriptor



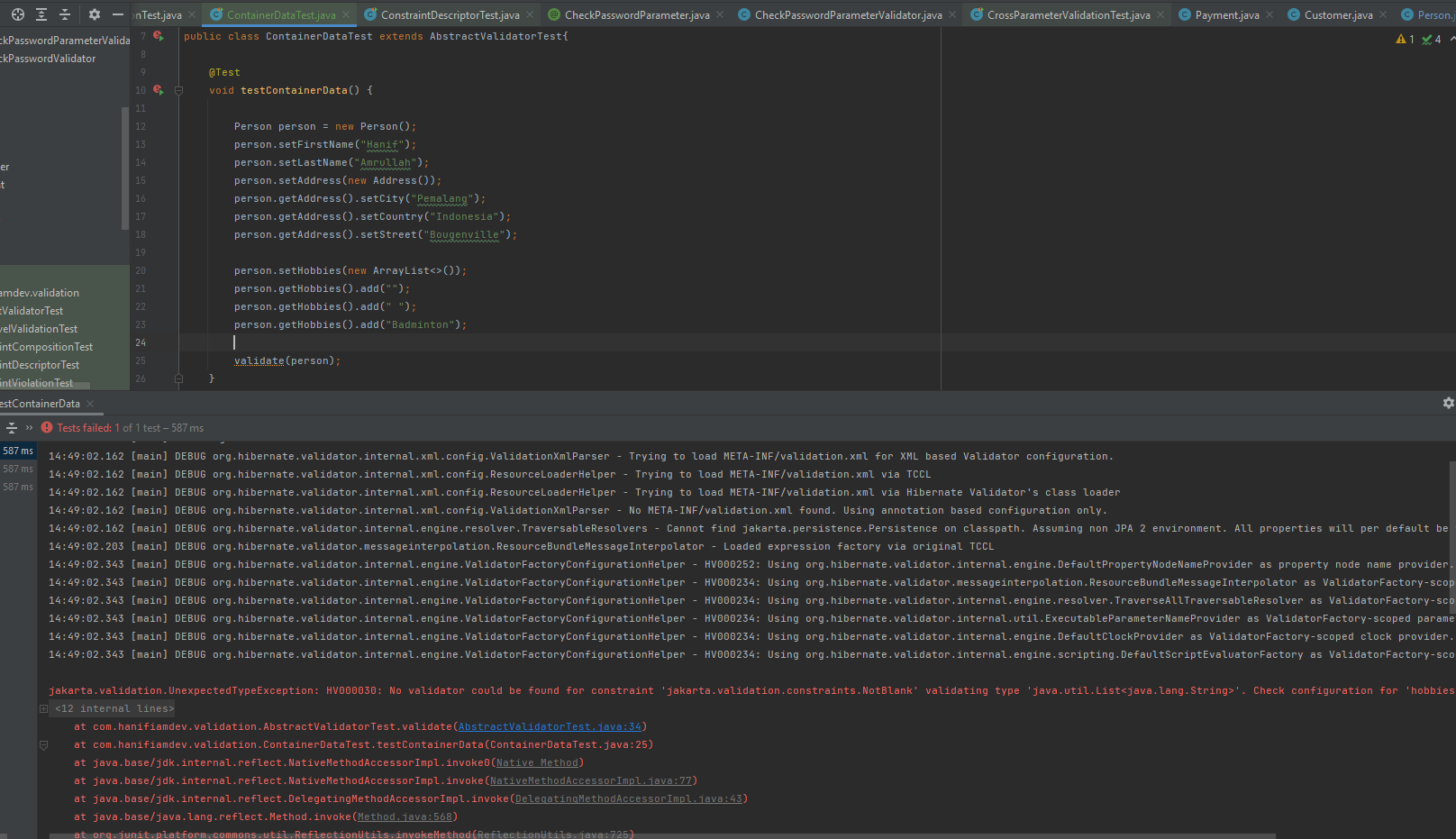
## **Container Data**

* Saat kita membuat class, kadang sering kita gunakan tipe data container, seperti misalnya Optional, Collection, List, Set, Map dan lain-lain
* Secara default, jika kita isi data tersebut dengan data Object, misal @Valid List<Address>, maka secara otomatis Bean Vallidation akan melakukan validasi semua data object Address sesuai dengan constraint yang ada di dalam class Address
* Tapi bagaimana jika kita memiliki misal field List<String>. atau Map<String, String>, bagaimana melakukan validasinya? Misal kita ingin semua data string di List tidak boleh kosong
* Untungnya, Bean Validation mendukung validasi terhadap data container seperti ini

Add Annotation Constraint validasi Not null didaladm container data yaitu misal List<String>



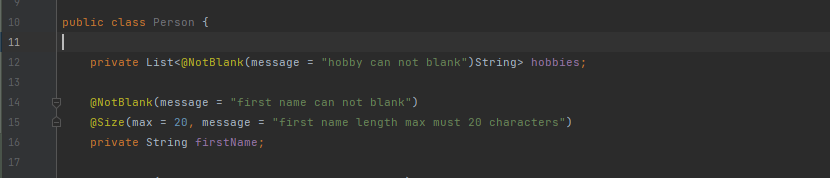
Test, maka unit test akan gagal karean container data tidak bisa langsung ditambahkan annotaion constraint seperti menambahkan @NotBlank di field hobbies



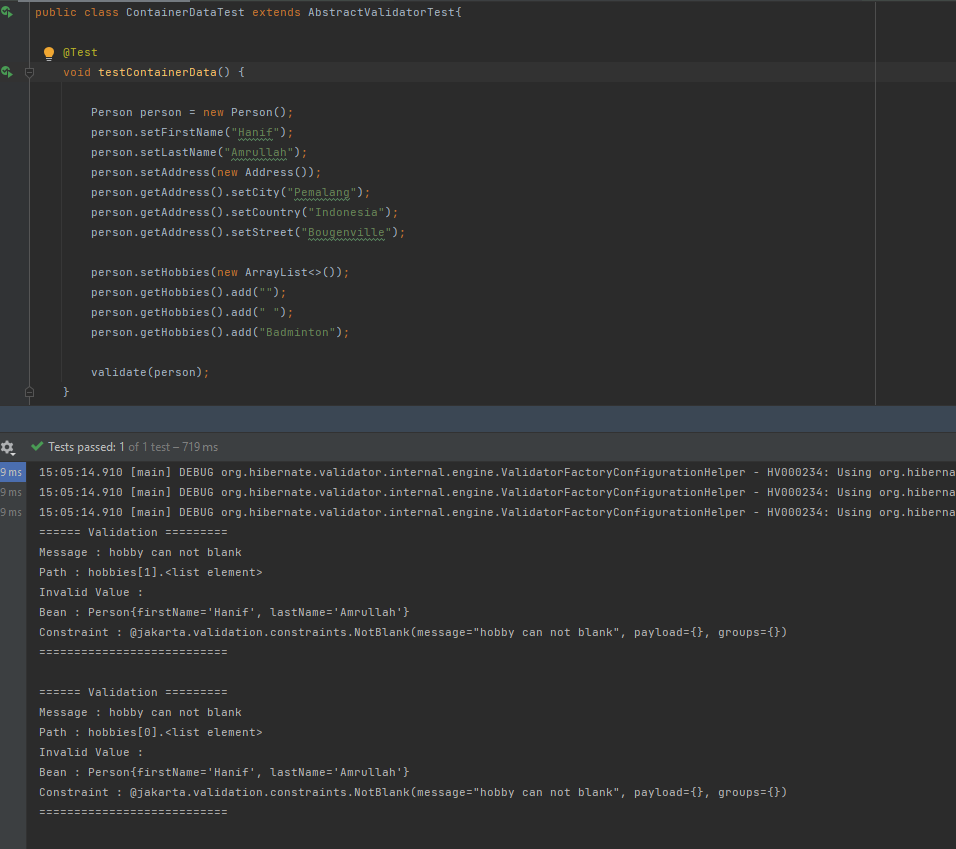
**Validate Container Data**

* Untuk melakukan validasi data jenis container Bean validation membutuhkan yang namanaya Value Extractor
* Value Extractor ini menjadikan Bean Validation bisa melakukan ekstraksi data dari container
* Secara default, Bean Validation mendukung semua data container yang tersedia di Java, seperti Optional, Collection, List, Iterable, Set dan Map
* Yang kita butuhkan, hanya dengan menambahkan Constraint pada generic type container tersebut

Ubah penempatan annotation constraintnya seperti berikut



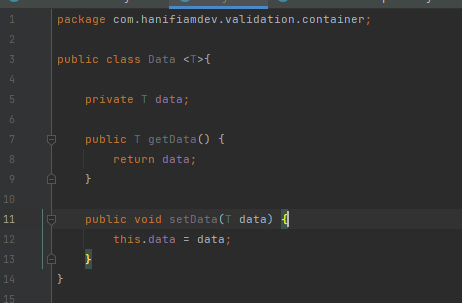
Unit Test Success



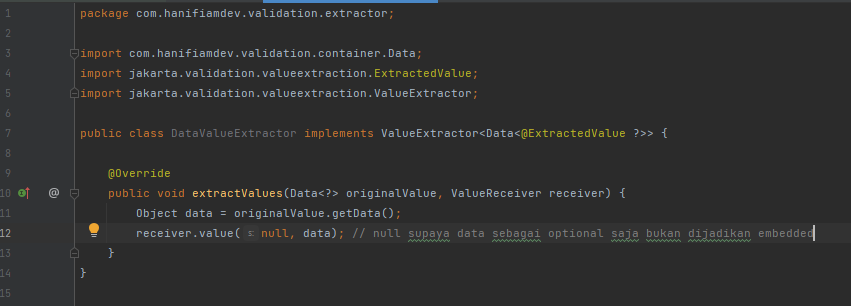
## **Value Extraction**

* Value Extraction meraupakan proses melakukan ekstraksi nilai dari data jenis container (kumpulan data), sehingga nilai-nilai nya bisa di validasi
* Sebelumnya sudah dijelaskan bahwa secara default Bean Validation mendukung value extraction terdapat data container yang ada di Java
* Bagaimana jika kita misal menggunakan data container sendiri atau misal menggunakan library bukan bawaan Java? Maka secara otomatis Bean Validation tidak bisa melakukan ekstrasi nilai yang terdapat di container, perlu kita lakukan secara manual
* Cara untuk memberi tahun Bean Validation cara melakukan ekstraksi, adalah dengan cara membuat ValueExtractor sendiri
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/valueextraction/valueextractor>

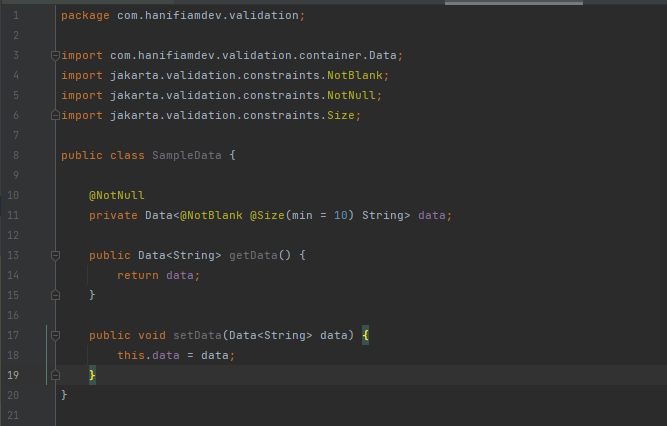
kode : Data Class



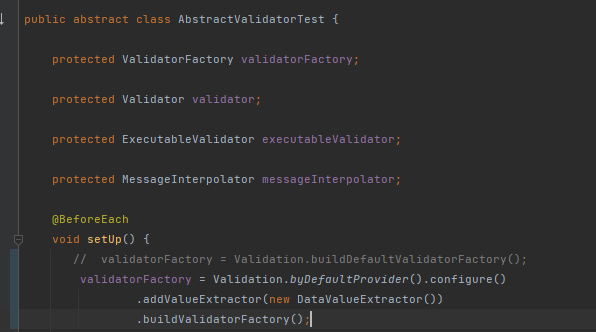
kode : Data Value Extractor



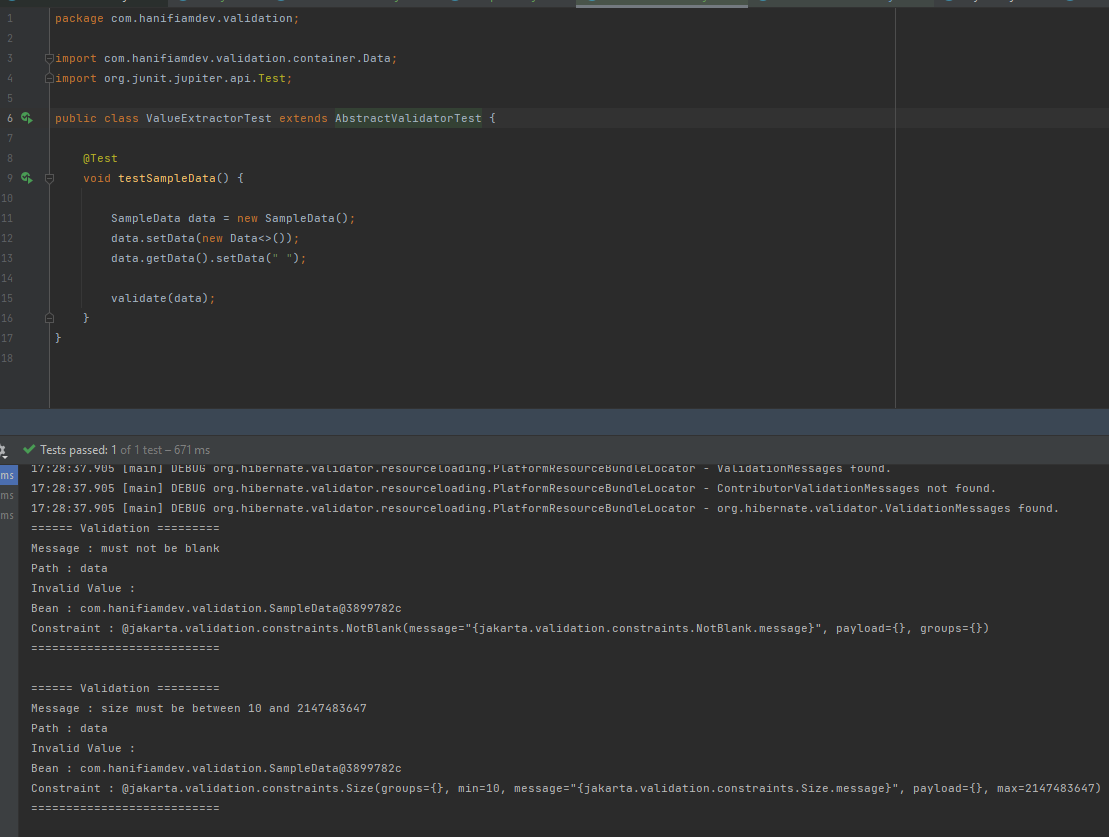
Kode : Sample Data Class



Ubah inisialisasi validatorfactory



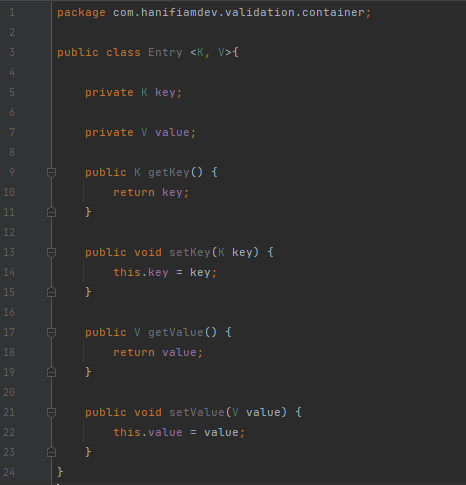
Test Validasi Generic Type



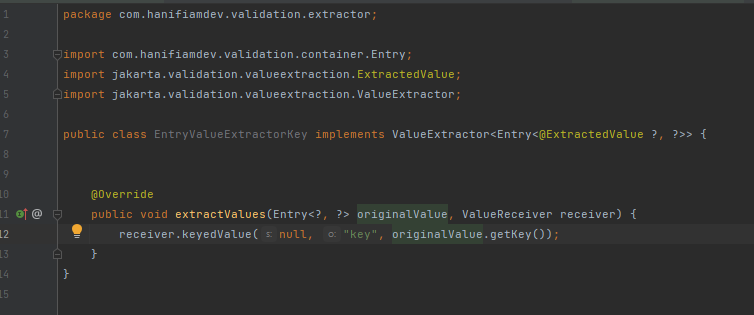
**Mutiple Generic Parameter Type**

* Secara default, saat kita membuat Value Extractor, annotation @Extracted hanya bisa digunakan satu kali
* Oleh karena itu, jika kita membuat container class generic yang menggunakan lebih dari satu generic parameter type, maka kita harus membuat Value Extractor nya sebanyak jumlah generic parameter type nya

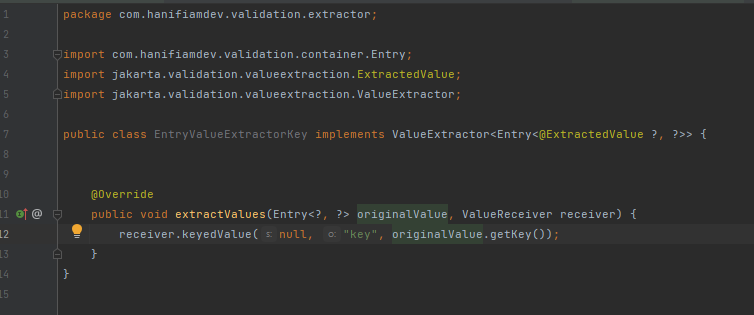
Kode : Entry Class ( punya dua geric Type K dan V)



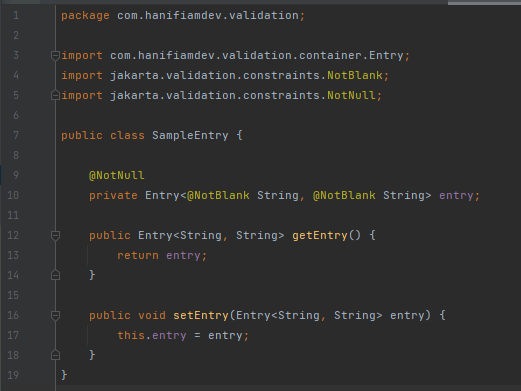
Kode : Entry Value Extractor Key



Kode : Entry Value Extractor Value



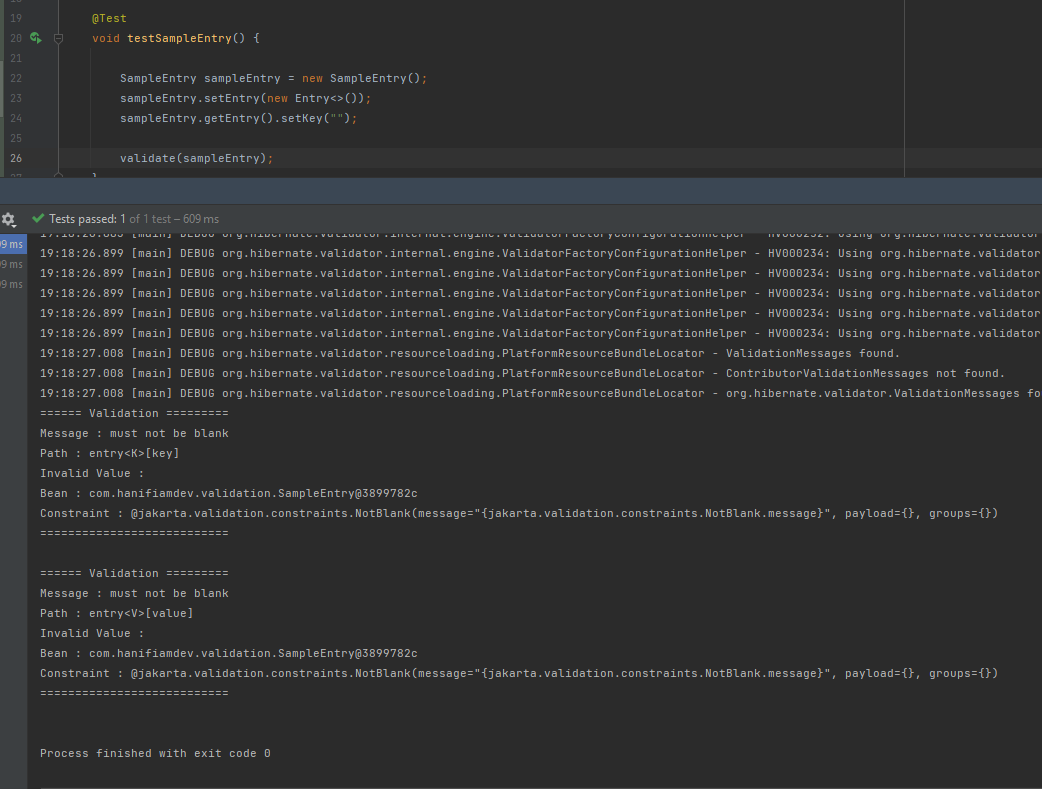
Kode : implementasi Value Extractor



Kode : registrasi validator factory



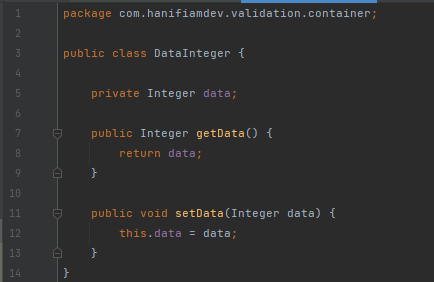
Kode : Unit Test



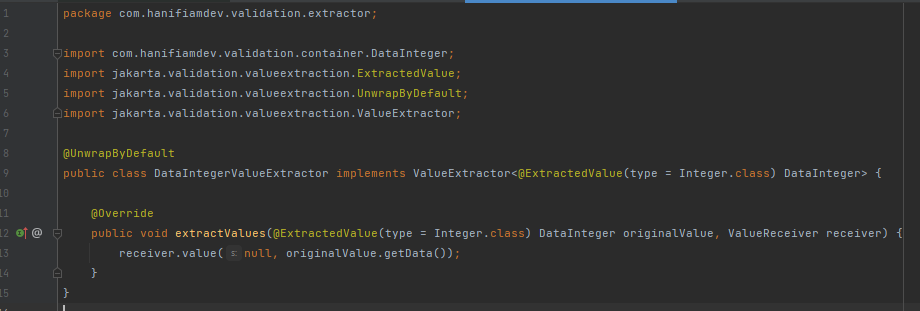
**Container Non Generic**

* Rata-rata, biasanya data container itu adalah class generic, namun beberapa kasus mungkin ada juga container yang bukan tipe generic
* Bagaimana untuk menangani hal ini? Untuk menangani hal ini, kita tetap bisa menggunakan @ExtractedValue, namun kita memberi tahu tipe data nilai dari container nya

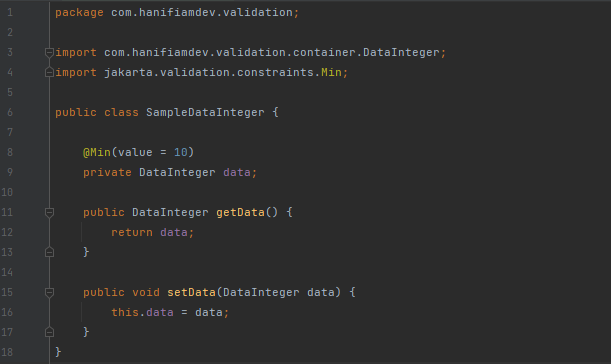
Kode : Data Integer



Kode : Data Integer Value Extractor



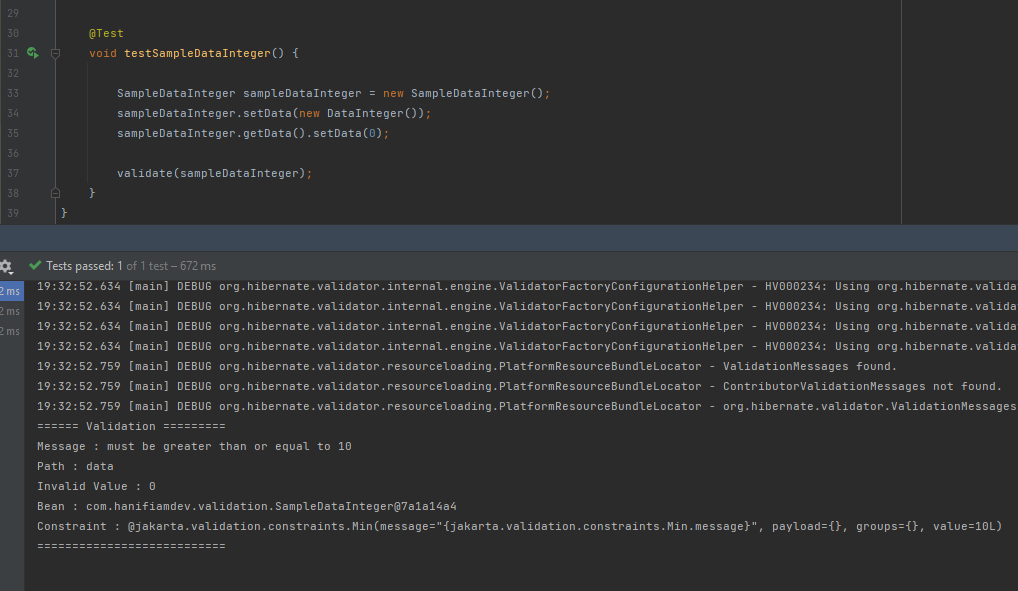
Kode : Implement validasi



Kode : Registrasi validator



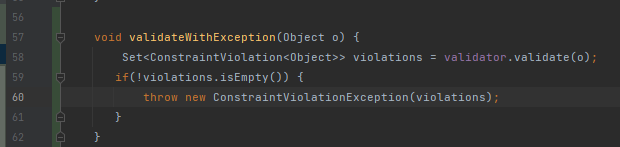
Kode : Unit Test



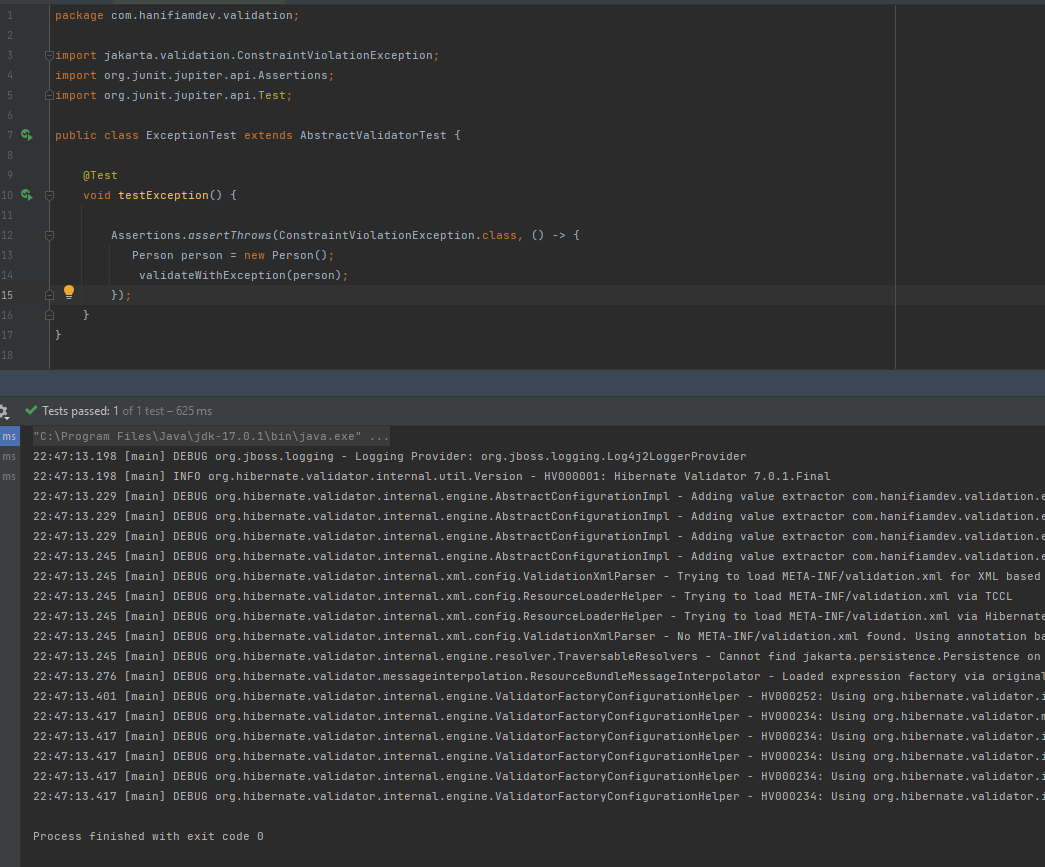
## **Constraint Violation Exception**

* Bean Validation secara default tidak membuat Exception terjadi validasi error
* Bean Validation hanya mengembalikan error validasi dalam bentuk Set berisi Constraint Violation
* Beberapa framework atau library, kadang menginginkan Exception jika terjadi validasi error
* Kita tidak butuh membuat exception secara manual, Bean Validation sudah menyediakannya, yaitu ConstraintViolationException
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/constraintviolationexception>

Kode : Constraint Violation Exception di kelas AbstractValidatorTest.java



Test Exception



## **Metadata**

* Materi ini sebenarnya bisa kita tangani dengan menggunakan Reflection
* Namun Bean Validation memberikan fitur untuk mempermudah kita
* Bean Validation memungkinkan kita melihat sebuah constraint yang terdapat di sebuah class
* Kita bisa melihat detail dari constraint tersebut, mirip seperti ketika kita melihat semua struktur class menggunakan Java Reflection
* Metadata untuk constraint, disimpan dalam object BeanDescriptor
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/metadata/beandescriptor>

