**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

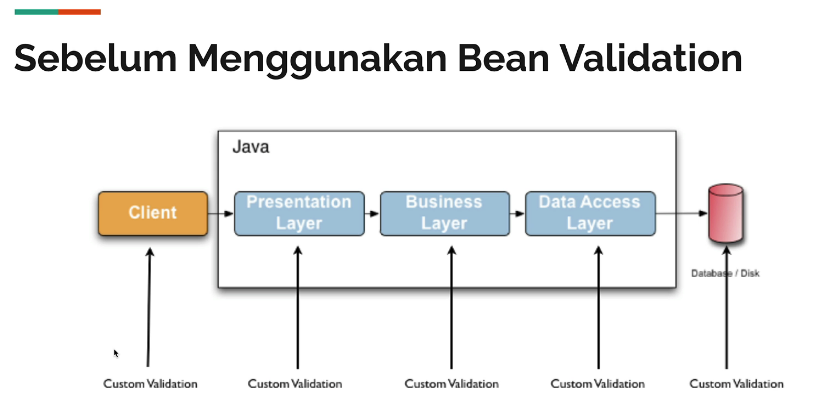
## **Pengenalan Bean Validation**

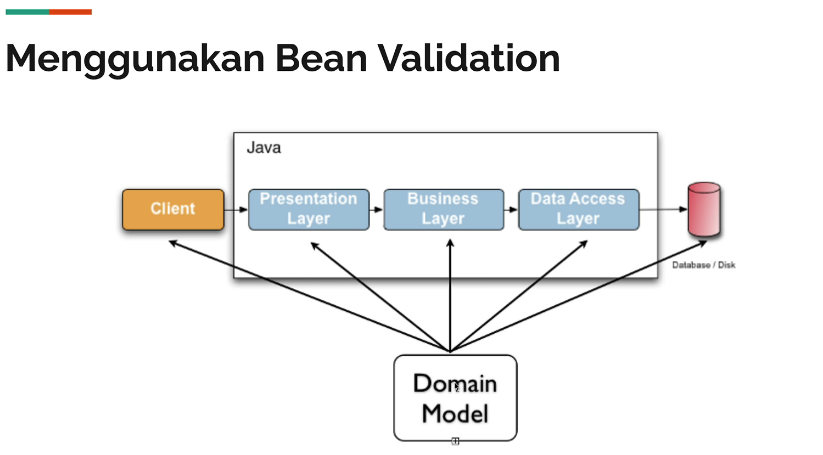
**Bean Validation**

* 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**

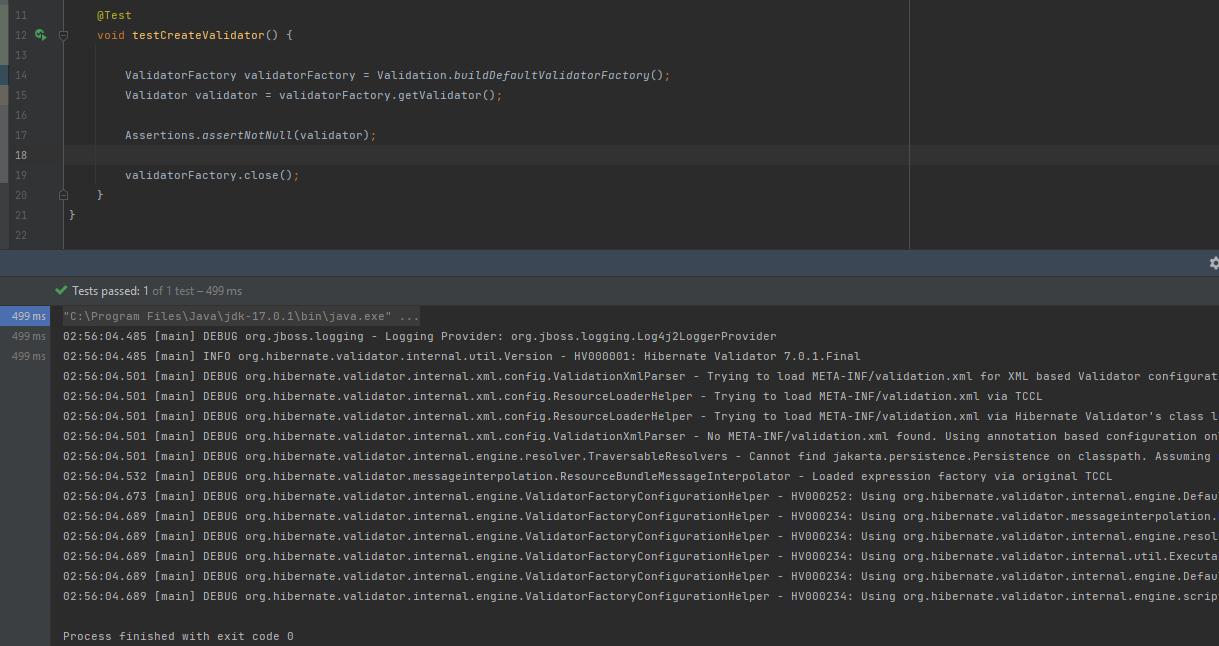
* Saat ini versi terbaru dari Bean validation adalah versi 3, sedangkan masih banyak yang menggunakan Bean Validation versi 2
* Bean Validation versi 2 masih menggunakan nama Java Enterprise, sedangkan Bean Validation versi 3 sudah menggunakan nama Jakarta
* Oleh karena itu saat ini, banyak package untuk 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**

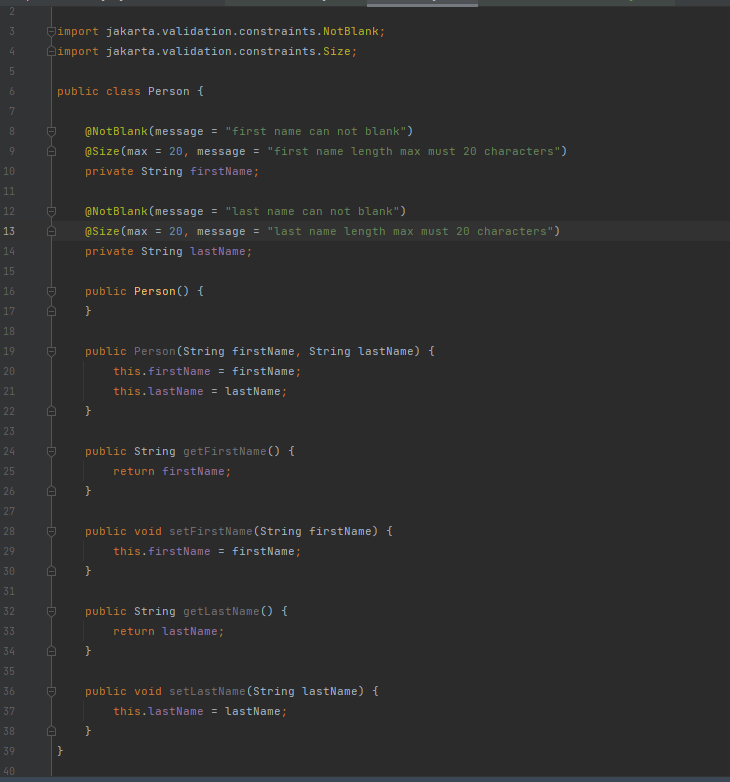
* Validator adalah class utama dalam Bean Validation
* Validator digunakan sebagai object untuk mengeksekusi validation
* Validator adalah object yang berat, oleh karena itu sebaiknya hanya dibuat satu kali saja dalam aplikasi
* <https://jakarta.ee/specifications/bean-validation/3.0/apidocs/jakarta/validation/validator>

**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**

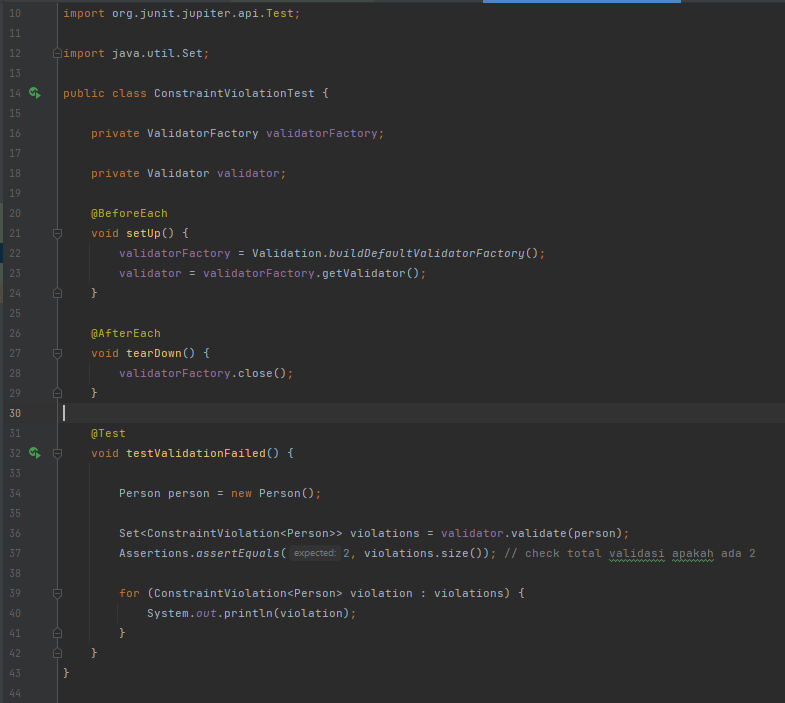
* Constraint merupakan Annotation yang digunakan sebagai penanda untuk target yang kita tambahkan (misal Field, Method, dan lain-lain)
* Bean Validation sudah menyediakan banayak sekali Constraint yang bisa langsung kita gunakan
* Jika kita butuh validasi yang berbeda, kita juga bisa membuat constraint 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**

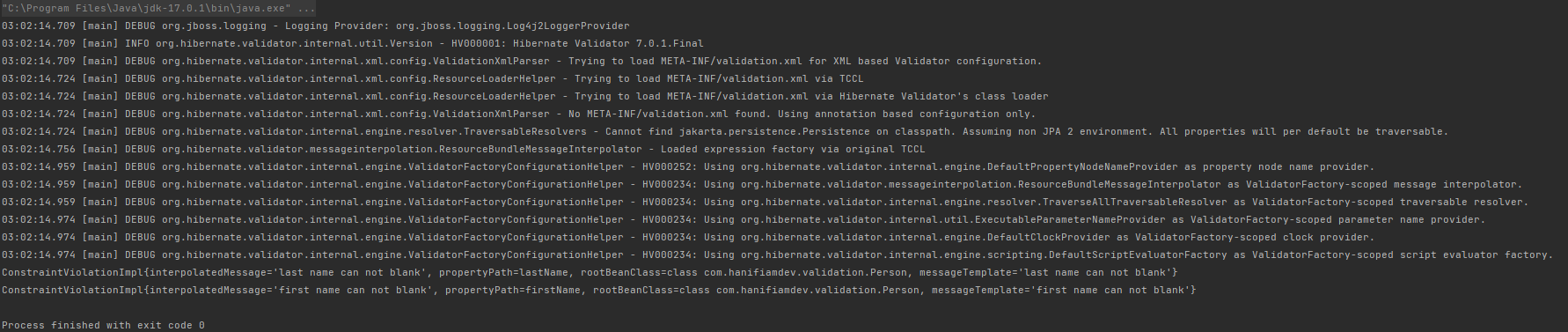
**Constraint Violation**

* 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



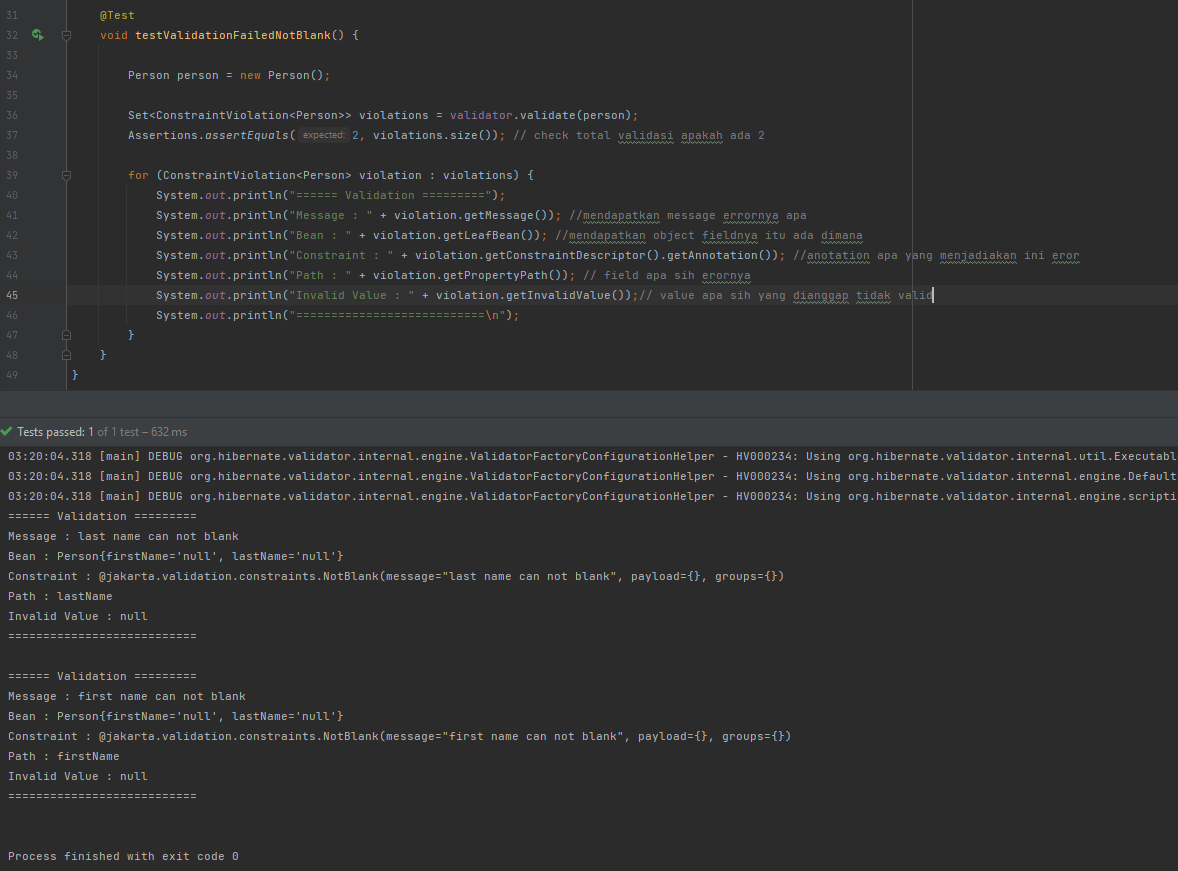
Result : Constraint Violation



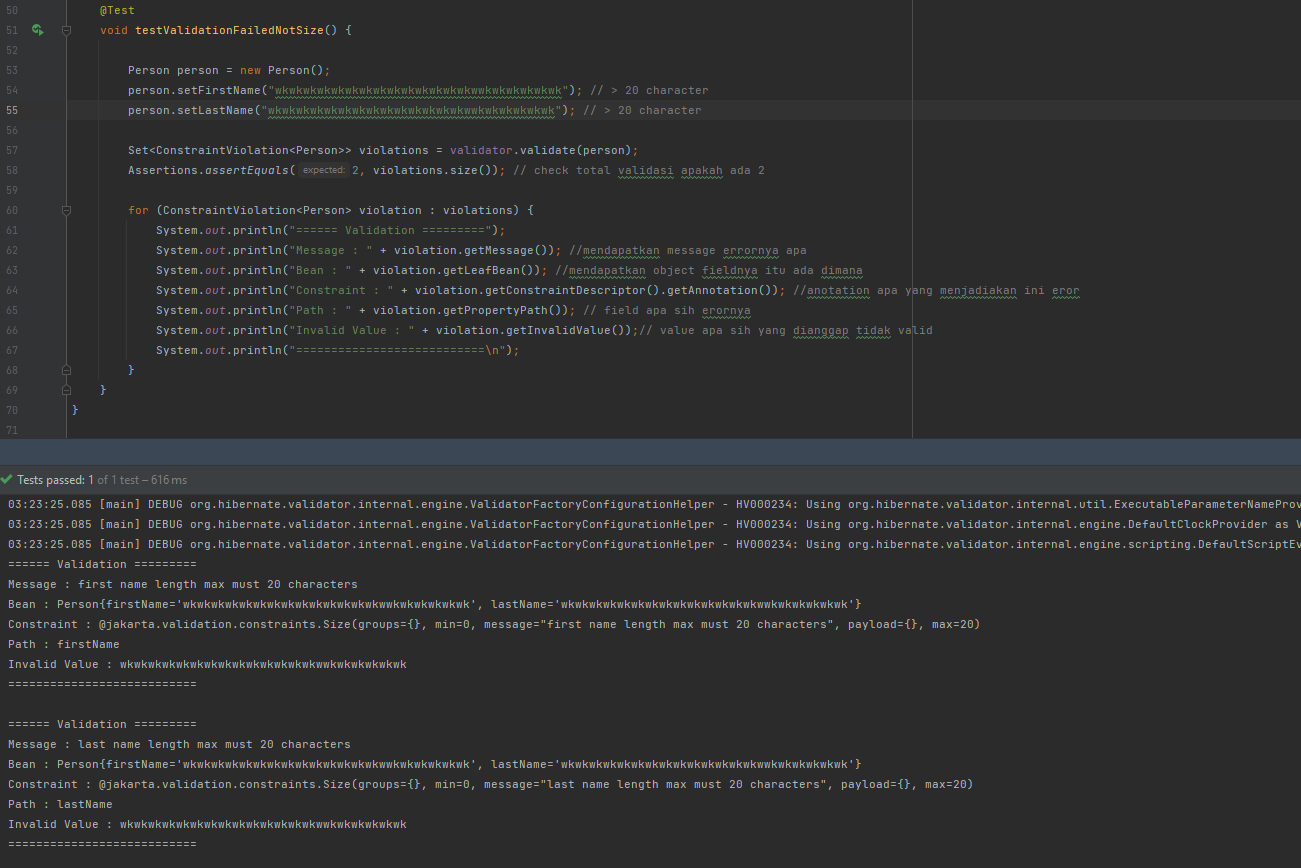
**Object Metadata**

* 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

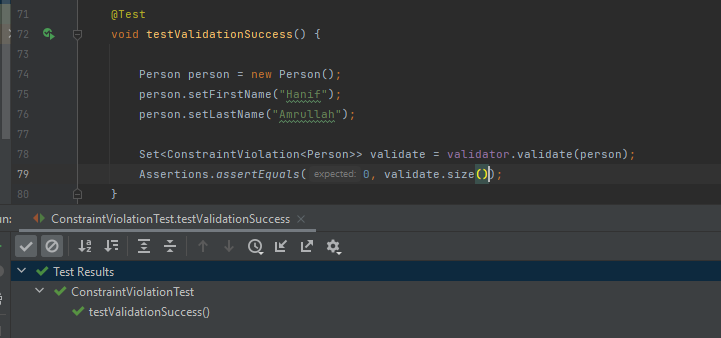
Test validasi Not Blank



Test validasi Size



Test Sukses validasi



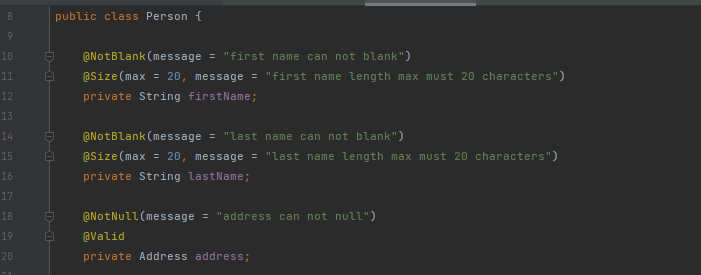
## **Nested Validation**

* 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>

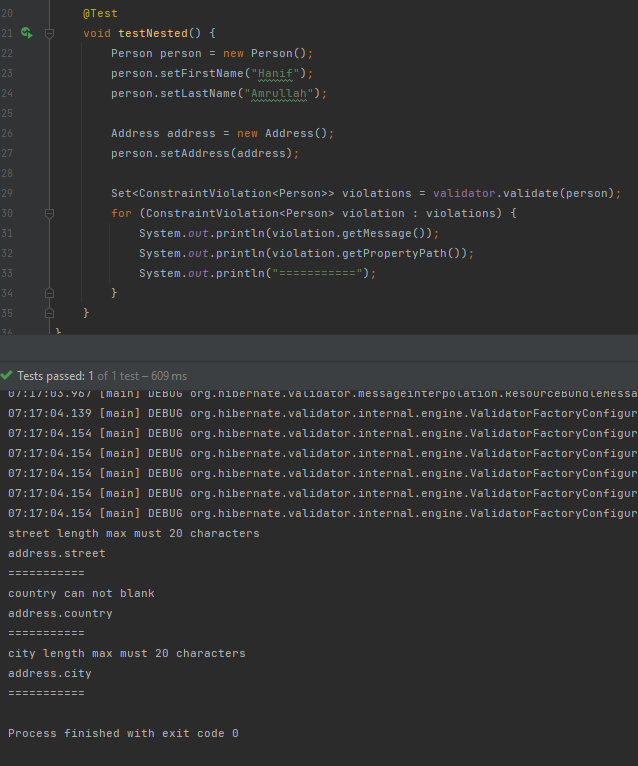
kode : Class Address



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



Address ikut divalidasi karena di kelas person sudah ditambahkan @Valid



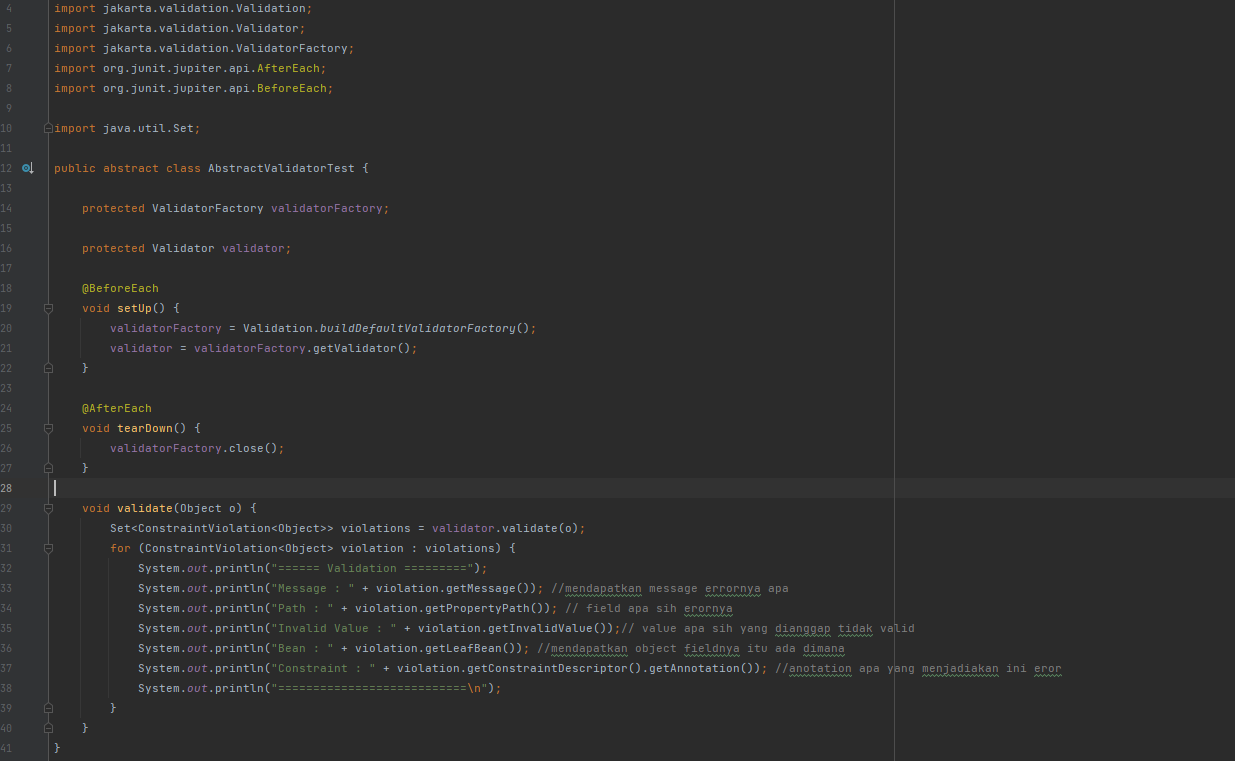
## **Hibrenate Validator Constraint**

* 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>

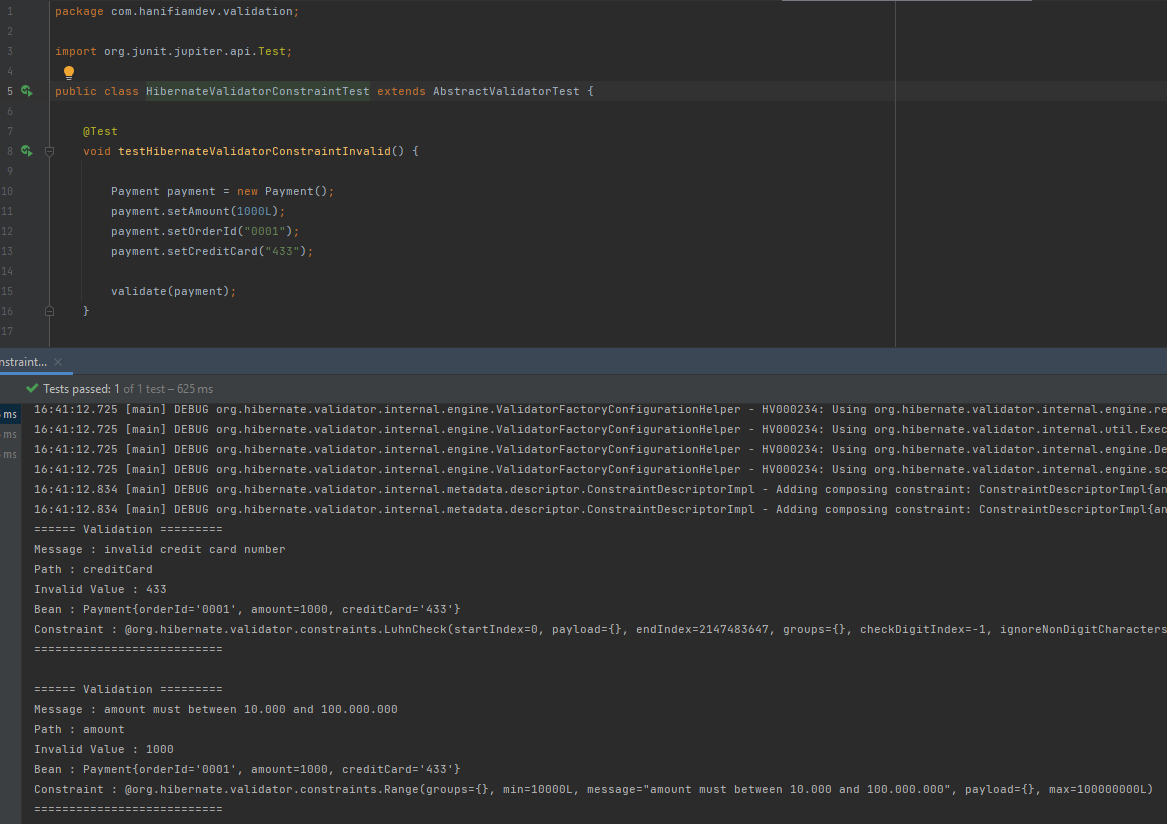
Payment Class



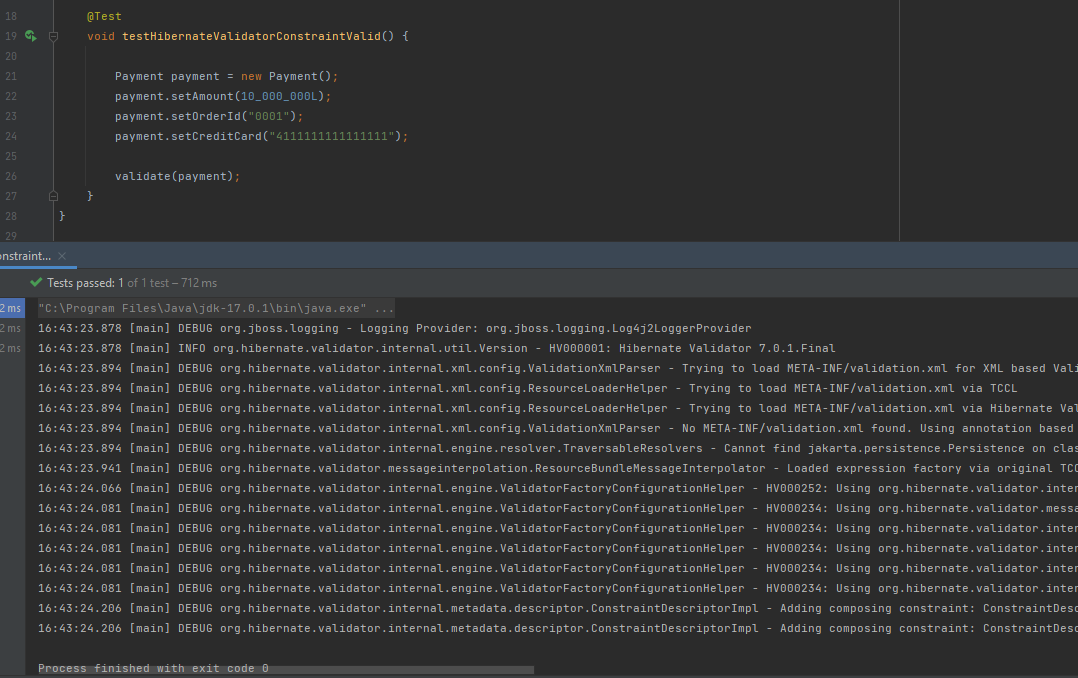
Buat kelas abstract supaya inisialisasi validator lebih mudah dan reusable



Test Invalid



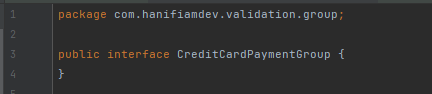
Test Valid

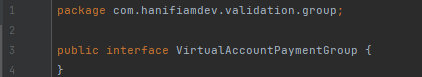


**Group Constraint**

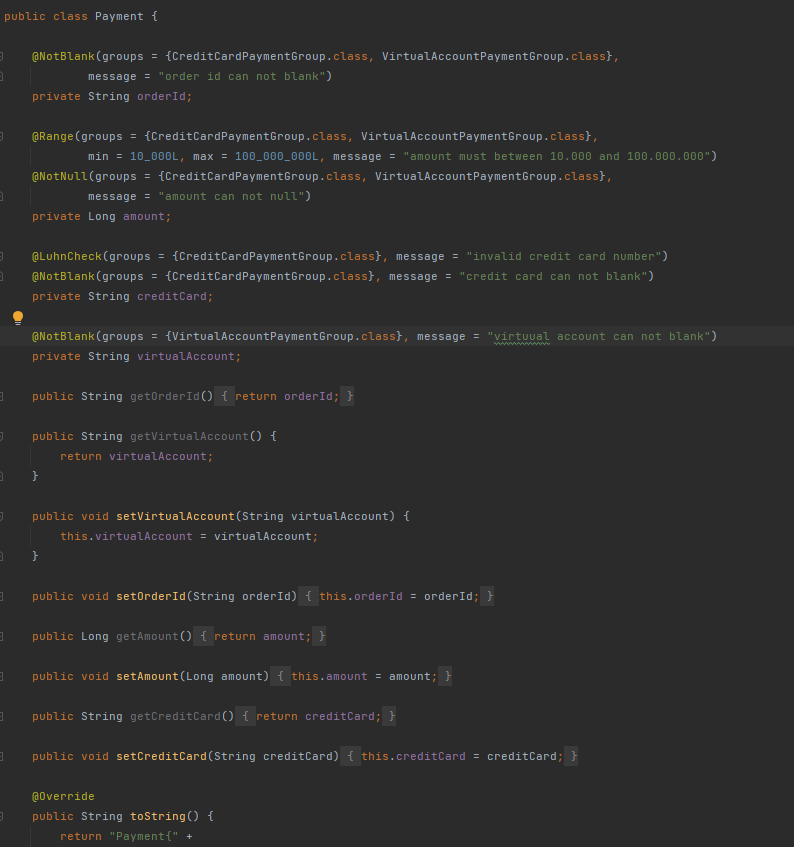
* Secara default, saat Validator melakukan validasi, validator akan memvalidasi semua field yang memiliki constraint
* Kadang, ada saatnya misal kita ingin melakukan pengecekan beberapa hal saja pada kondisi tertentu, misal terdapat satu class yang digunakan untuk beberapa aksi,sehingga membutuhkan kombinasi 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

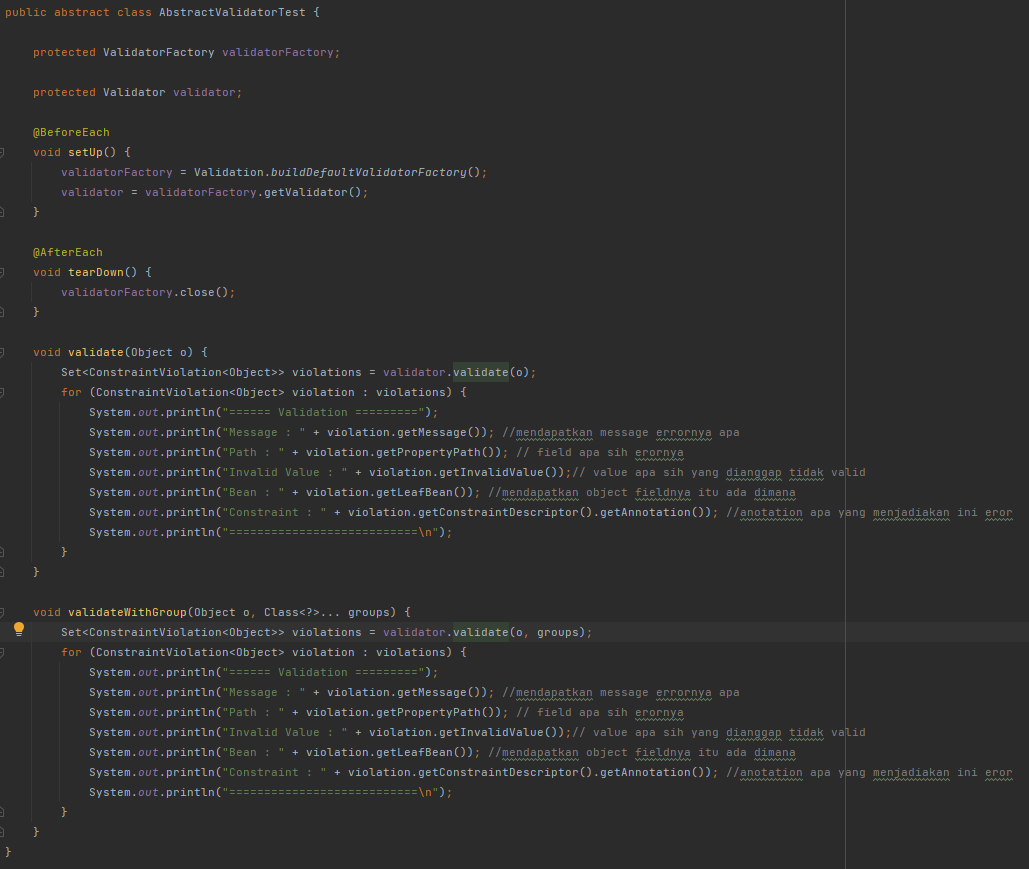




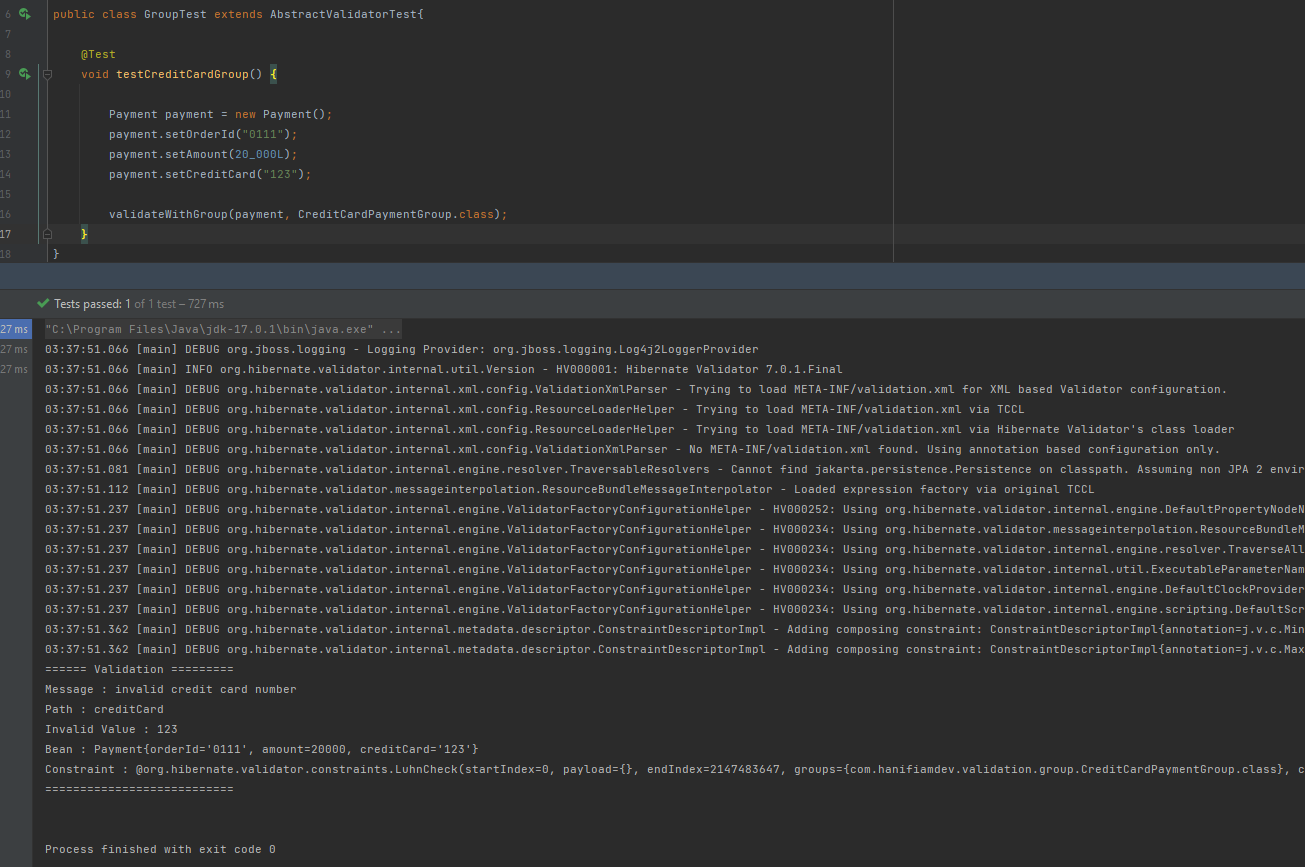
Add group di dalam bean validation, menambahkan field cirtualAccount beserta getter and setter



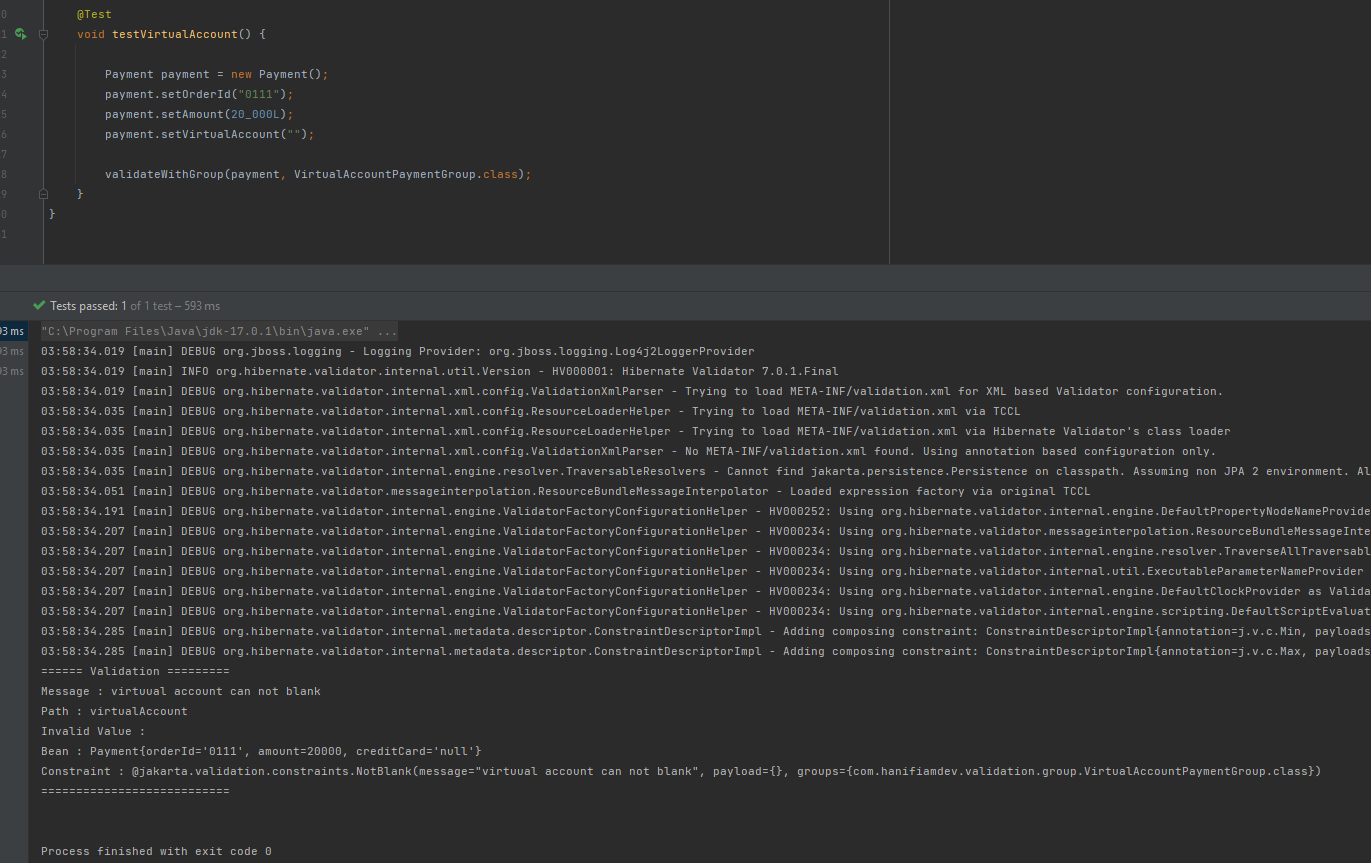
Sesuaikan kelas AbstractValidatorTest supaya bisa eksekusi dengan input param group



Test Validasi terhadap group CreditCardPaymentGroup.class



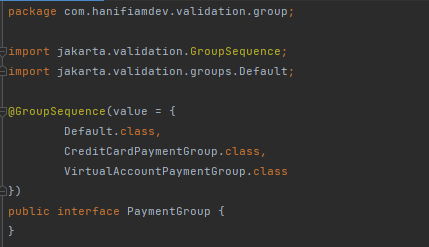
Test Validasi terhadap group CreditCardPaymentGroup.class

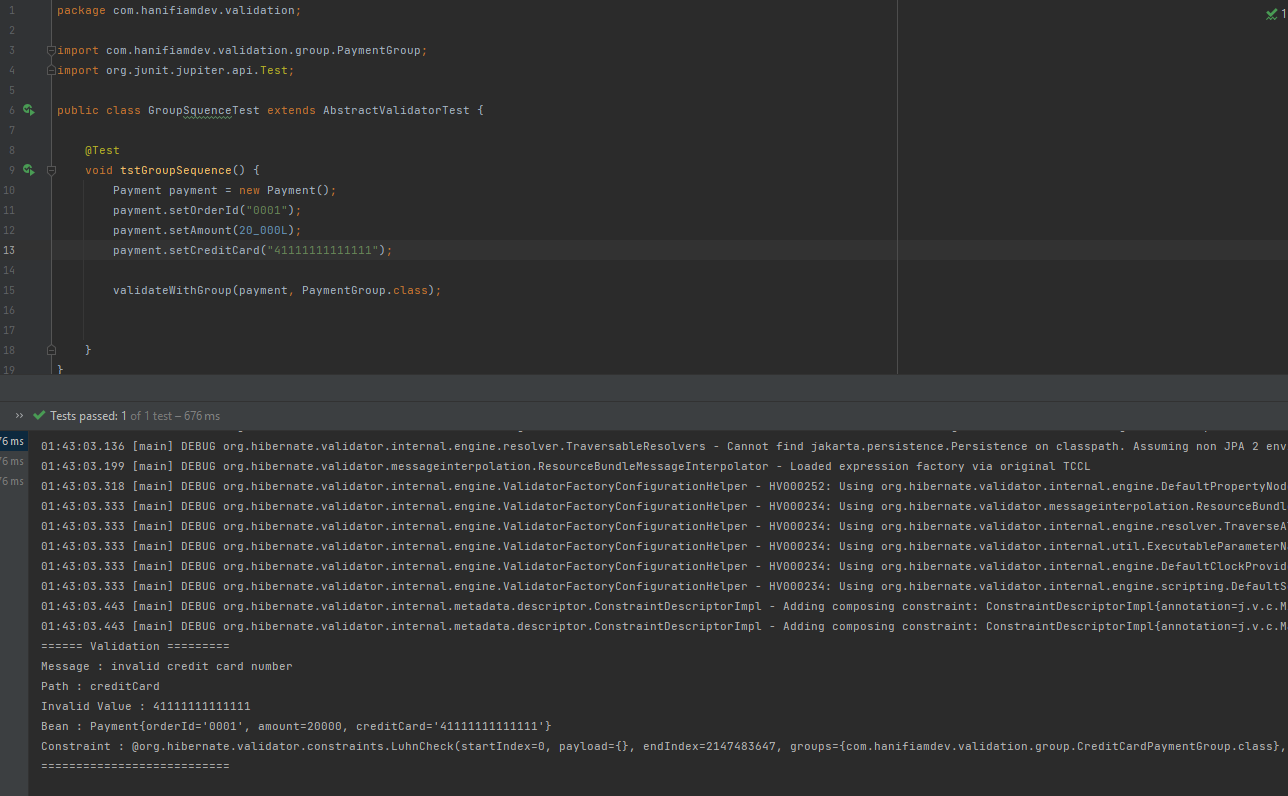


## **Group Sequence**

* 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





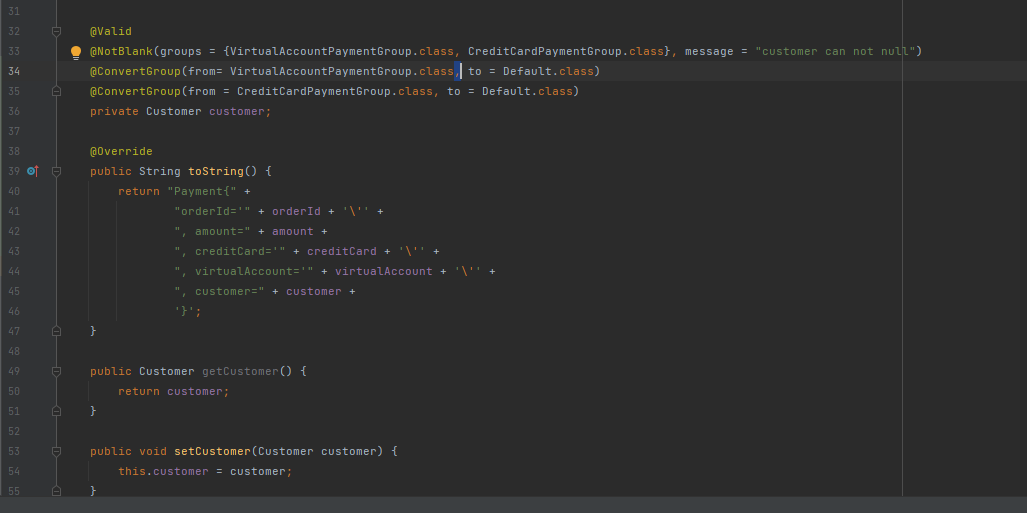
## **Group Conversion**

* 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

