Validation in Scala

1. Validations of members of case class, one validation per member - covered by Daniela Sfregola in her Scala Exchange 2016 talk.
2. Multiple validations of a single item.
3. Combining collections of validations
4. Combining validations of non-related things - just to know if there is anything wrong.
5. Combine collection of validations into a collection of values if they are all valid or a single combined validation if at least one is invalid.

2. Multiple validations of a single item.

**package** com.ce  
  
**import** cats.Semigroup  
**import** cats.data.\_  
**import** cats.implicits.\_  
**import** com.ce.validation.{Err, ErrorCode, Validation}  
  
**object** CombineImprovedValidator **extends** App {  
 **implicit val** *nonCombiningStringSemigroup* = *Semigroup*(*NonCombiningString*(""))  
  
 **def** validateEmailByRegex(email: NonCombiningString): Validation[NonCombiningString] = {  
 **val** emailRegex = """^[a-zA-Z0-9\.!#$%&'\*+/=?^\_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)\*$""".r  
 email.value **match** {  
 **case** emailRegex(e) => Validated.valid(*NonCombiningString*(e))  
 **case** \_ => Validated.invalid(*List*(*Err*(ErrorCode.*InvalidEmailFormat*, "invalid email format")))  
 }  
 }  
  
 **def** validateEmailByKeyword(email: NonCombiningString, keyword:String): Validation[NonCombiningString] =  
 **if** (email.value.toLowerCase contains keyword) Validated.valid(email)  
 **else** Validated.invalid(*List*(*Err*(ErrorCode.*EmailMustContainWordGood*,  
 s"email must contain keyword **$**{keyword}")))  
  
 **def** validatePhoneByRegex(phone: NonCombiningString): Validation[NonCombiningString] = {  
 **val** phoneRegex = """^\+(?:[0-9] ?){6,14}[0-9]$""".r  
 phone.value **match** {  
 **case** phoneRegex(p) => Validated.valid(*NonCombiningString*(p))  
 **case** \_ => Validated.invalid(*List*(*Err*(ErrorCode.*PhoneMustBeNumeric*,  
 s"invalid phone number format")))  
 }  
 }  
  
 **def** validatePhoneByPrefix(phone: NonCombiningString, prefix:String): Validation[NonCombiningString] =  
 **if** (phone.value contains prefix) Validated.valid(phone)  
 **else** Validated.invalid(*List*(*Err*(ErrorCode.*PhoneMustHaveUKCountryCode*,  
 s"phone must have prefix: **$**{prefix}")))  
  
 **def** validateData(d: MyData): Validation[MyData] = {  
 **val** validEmail = *validateEmailByRegex*(d.email)  
 .combine(*validateEmailByKeyword*(d.email, "good"))  
  
 **val** validPhone = *validatePhoneByRegex*(d.phone)  
 .combine(*validatePhoneByPrefix*(d.phone, "+44"))  
  
 (validEmail |@| validPhone).map(MyData)  
 }  
  
 **val** *v* = *validateData*(*MyData*(*NonCombiningString*("wrong email"),  
 *NonCombiningString*("wrong phone number")))  
 *v*.leftMap{ e =>  
 e.foreach(ee => *println*(ee.msg))  
 }  
}

Gives the following output:

invalid email format

email must contain keyword good

invalid phone number format

phone must have prefix: +44

Where:

**case class** MyData(email: NonCombiningString, phone: NonCombiningString)

and:

**case class** NonCombiningString(value:String) **extends** AnyVal **with** Semigroup[NonCombiningString] {  
 **override def** combine(x: NonCombiningString, y: NonCombiningString): NonCombiningString = x  
}

Question – is there a better way, how to avoid the need for NonCombiningString?

3. Combining collections of validations.

**def** validateData(d: MyData): Validation[MyData] = {  
 **val** emailValidations = *List*(*validateEmailByRegex*(d.email), *validateEmailByKeyword*(d.email, "good"))  
  
 **val** phoneValidations = *List*(*validatePhoneByRegex*(d.phone), *validatePhoneByPrefix*(d.phone, "+44"))  
  
 **val** validEmail = emailValidations.reduceLeft(\_ combine \_)  
 **val** validPhone = phoneValidations.reduceLeft(\_ combine \_)  
  
 (validEmail |@| validPhone).map(MyData)  
}  
  
**val** *v* = *validateData*(  
 *MyData*(*NonCombiningString*("wrong email"), *NonCombiningString*("wrong phone number"))  
)  
*v*.leftMap{ e =>  
 e.foreach(ee => *println*(ee.msg))  
}

Same output as before

4. Combining validations of non-related items – just to know if anything is wrong.

We need a combine version that would be agnostic to value type of the validation.

TBD.

5. Convert collection of validations into a collection of values if the are all valid or a single combined failed validation if one or more are invalid.

TBD.