// kwestia - validations of members of case class, one validation per member - covered by Daniela

// kwestia - multiple validations of a single thing

// pozostaje kwestia - combining collections of validations

// jeszcze jedna - combining validations of non-related things - just to know if there is anything wrong

// jeszcze jedna - finding valid in a collection of validations, otherwise combine invalids

**case class** Kubek(text:String) **extends** Semigroup[Kubek] {  
 **override def** combine(x: Kubek, y: Kubek): Kubek = x  
}  
  
**case class** Kubek2(s1:String, i2:Int)  
  
**object** KubekValidator {  
 **def** doVal1(k:Kubek): Validation[Kubek] = Validated.invalid(*List*(*RulesFileError*(1,"val1")))  
 **def** doVal2(k:Kubek): Validation[Kubek] = Validated.invalid(*List*(*RulesFileError*(2,"val2")))  
 **def** doVal3(k:Kubek): Validation[Kubek] = Validated.valid(k)  
 **def** doVal4(k:Kubek): Validation[Kubek] = Validated.valid(k)  
 **def** doVal5(s:String): Validation[String] = Validated.invalid(*List*(*RulesFileError*(5,"val5")))  
 **def** doVal6(s:String): Validation[String] = Validated.invalid(*List*(*RulesFileError*(6,"val6")))  
 **def** doVal7(s:String): Validation[String] = Validated.valid(s)  
 **def** doVal8(s:String): Validation[String] = Validated.valid(s)  
}  
  
**object** StringSemi **extends** Semigroup[String] {  
 **override def** combine(x: String, y: String): String = x  
}  
  
**object** KubelTest **extends** App {  
 **import** KubekValidator.\_  
 **implicit val** *errorListSemigroup* = *Semigroup*(ErrorListSemigroup)  
 **implicit val** *kubekSemigroup* = *Semigroup*(*Kubek*(""))  
 **implicit val** *stringSemigroup* = *Semigroup*(StringSemi)  
  
 **val** *k* = *Kubek*("abc")  
  
// val v1 = doVal3(k)  
// val v2 = doVal4(k)  
 **val** *v1* = *doVal7*("77")  
 **val** *v2* = *doVal8*("88")  
  
 *println*(*v1* combine *v2*)  
 *println*(*v1* product *v2*)  
  
 **def** fff(a:String, b:Int)(f:(String,Int) => Kubek2) = {  
 f(a,b).toString  
 }  
  
 *println*(*fff*("a",5)(Kubek2))  
  
 // kwestia - validations of members of case class, one validation per member - covered by Daniela  
  
 // kwestia - multiple validations of a single thing  
  
 // pozostaje kwestia - combining collections of validations  
  
 // jeszcze jedna - combining validations of non-related things - just to know if there is anything wrong  
}

// kwestia - multiple validations of a single thing

**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?

// kwestia - 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