Exercice 12 :

1. Explorons les avantages de l'utilisation scala.util.{Try,Success,Failure}

Le trio de classes nommé `Try`, `Success`, et `Failure` fonctionne exactement comme `Option`, `Some`, et `None`, mais avec deux fonctionnalités intéressantes :

- `Try` rend très simple la détection des exceptions

- `Failure` contient l'exception

Nous allons nous baser ici, sur un exemple avec la methode toInt()

```
brice@wambdevps:~$ scala
Welcome to Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 11.0.16).
Type in expressions for evaluation. Or try :help.

scala> :paste
// Entering paste mode (ctrl-D to finish)

//Tout d'abord, importez les classes dans la portée
import scala.util.{Try,Success,Failure}
def toInt(s: String): Try[Int] = Try {
    Integer.parseInt(s.trim)
}

// Exiting paste mode, now interpreting.

import scala.util.{Try, Success, Failure}
toInt: (s: String)scala.util.Try[Int]

scala> val a = toInt("5")
a: scala.util.Try[Int] = Success(5)
```

Deuxièmement, voici à quoi cela ressemble lorsqu'une `Integer.parseInt` exception est lancée :

```
scala> val a = toInt("bigdata")
a: scala.util.Try[Int] = Failure(java.lang.NumberFormatException: For input string: "bigdata")

scala>
```