Exercises, SDJ1

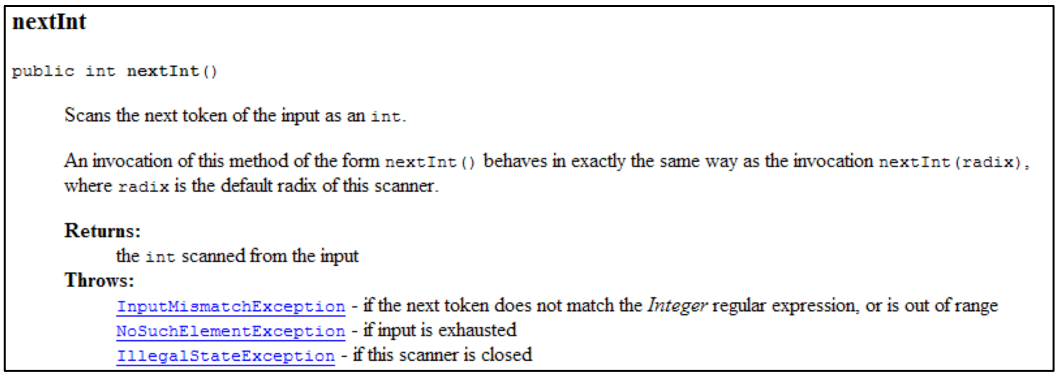
Exercise: Validation and try-catch

Write a class with a `main` method, in which you read 5 two-digit integers (i.e. in the range of 10 through 99, both inclusive) from keyboard and store these integers in an array. You have to use a validation check in your loop to avoid illegal values ending up with an array with exactly 5 two-digit integers.

If you use the method `nextInt()` in the `Scanner` class this method will throw an exception of type `java.util.InputMismatchException` if the keyboard input cannot be parsed to an integer (e.g. if it is the word "`hello`"). Use a try-catch block as part of your code in order to avoid program termination if the keyboard input is not an integer. See e.g. documentation for the Scanner method `nextInt()` below

Sample run (**bold red** values are keyboard input):

```
Type an integer in the range 10-99: 7
The input is not an integer in the range 10-99, try again
Type an integer in the range 10-99: 32
Type an integer in the range 10-99: 21
Type an integer in the range 10-99: hello
The input is not an integer in the range 10-99, try again
Type an integer in the range 10-99: 58
Type an integer in the range 10-99: 88
Type an integer in the range 10-99: 79
Program successfully ended
Values stored: 32, 21, 58, 88, 79
```



nextInt

`public int nextInt()`

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:
the `int` scanned from the input

Throws:
`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range
`NoSuchElementException` - if input is exhausted
`IllegalStateException` - if this scanner is closed

Exercise: Validation, try-catch and terminating the loop

Write a class with a `main` method in which you read an unspecified number of integers from keyboard, calculate and print out the sum. To indicate the end of inputs use the string "`end`". Omit any other input strings, which cannot be parsed to an integer. Like in the previous exercise, you could use a try-catch block to catch exceptions of type `java.util.InputMismatchException`
Hint: store the (wrong) input string in the catch block (using method nextLine) and check if this is equal to "end"

Sample run (bold red values are keyboard input):

```
Type an integer: 7
Type an integer: -3
Type an integer: hello
The input is not an integer, try again
Type an integer: 0
Type an integer: 12
Type an integer: end
The sum of the 4 integers is 16
```