IOOP2 2025 Lab Sheet A
Data Validation

This Lab Sheet contains material based on Week 1 topics.

The deadline for Moodle submission of this lab exercise is 12:00 noon on Thursday 2 October. No extensions are possible.

Aims and objectives

- Writing Java code to solve problems using primitive types, strings, arrays, loops, and conditionals
- Using built-in Java classes and methods to process data
- Showing intermediate steps and explaining your code clearly

Starter code

First, download the file **LabA.zip** from the Moodle page and unzip it somewhere in your own file space. Open the **LabA** folder in VSCode to access the starter code.

If you are not already familiar with the VSCode development environment, you should first follow the separate instructions to get started with VSCode.

This process can be a bit tricky if VSCode is not configured correctly, so please ask a demonstrator or tutor for help in the lab if it doesn't work.

Tasks

Both tasks this week involve various forms of data validation: checking that a barcode and an IBAN bank account identifier. **Note that the details of each task have deliberately been changed slightly from the standard version of the algorithm that you might see online.**

Each program should be written in a separate .java file and will be run independently. We have provided a starter version of each file in the **LabA** project where you should write your solution in the place indicated. Do not worry about the other syntax in these files (e.g., "import", "public class", "public static void main") – we will cover these aspects of Java in the weeks to come. For now, you just need to worry about writing your own code.

You should not modify the provided input code; only write your solution where indicated.

Hints and tips

Hint 1: The following methods of the **String** class may be useful for your implementations.[1] Assume that **`str`** is a variable of type **String**. Then …

- **`str.toCharArray()`** returns a **`char[]`** corresponding to the characters in **str**
- **`str.length()`** returns the length of **`str`** in characters
- **`str.charAt(i)`** returns the character at position **`i`** in **`str`**
- **`str.substring(i, j)`** returns a new **`String`** which is the set of characters beginning at index **`i`** and ending at index **`j-1`**

Hint 2: The following methods of the **Character** and **Integer** classes may also be useful. Assume that **`c`** is a variable of type **char** (and **str** is a **String** as before). Then …

- **`Character.isDigit(c)`** returns true if **`c`** is a digit (0-9) and false otherwise
- **`Character.isUpperCase(c)`** returns true if **`c`** is a capital letter (A-Z) and false otherwise
- **`Character.getNumericValue(c)`** converts a **`char`** value **`c`** into its corresponding integer value (e.g., from '5' to 5)
- **`Integer.valueOf(str)`** converts a **`String`** value **`str`** into its corresponding integer value (e.g., "123" into 123).

Task 1: EAN-13 Barcode validation

Your task is to implement a variation of the method to generate a check digit for an EAN-13. You can read more about barcodes at https://en.wikipedia.org/wiki/International_Article_Number; here is a summary:

An International Article Number (EAN) is a barcode used worldwide for marking products sold at retail point of sale. The most common form of an EAN is EAN-13, which is made up of 13 digits. The final digit of every EAN-13 is a **check digit**, which is used to verify that a barcode has been scanned properly – it is computed from the other 12 digits using a sum of products as described below. An EAN is only valid if the check digit is correct.

For this lab, we will use the following process to compute check digits: **note that this is slightly different from the standard process that you might read about online.**

1. Read the barcode prefix from the user (this code is provided).
2. Verify that the prefix is 12 characters long, and that all characters are digits.
    - If not, you should print "Invalid barcode prefix" instead of continuing further.
3. Each digit is alternately multiplied by 1 or 4. These products are summed modulo 10 to give a value ranging from 0 to 9.
4. The check digit is then computed by subtracting the result of the above calculation from 10. If the result is 10, then the digit is 0.

[1] Full documentation of the **String** class can be found at https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/String.html

You should implement this algorithm by completing the code in the **main** method in the **EANBarcodeCompleter.java** file.

Worked example

The following example (based on the Wikipedia page) should make the process easier to follow. For EAN-13 barcode prefix **400638133393?**, the check digit calculation is:

digits	**4**	**0**	**0**	**6**	**3**	**8**	**1**	**3**	**3**	**3**	**9**	**3**
weight	1	4	1	4	1	4	1	4	1	4	1	4
product	4	0	0	24	3	32	1	12	3	12	9	12
running total	4	4	4	28	31	63	64	76	79	91	100	112

The final sum is 112. 112 modulo 10 is 2. If we subtract this from 10, then we compute a check digit of 8, so the complete barcode is 4006381333938.

Sample output

Here are some examples of the code being run. Each of these segments shows a separate run, showing the content of the Terminal window, where user input is indicated in *italics*. Note: because we are multiplying by 4 (where the true EAN validation process uses 3), you should not validate examples with online barcode checkers, because they will mark our examples as invalid.

```
Enter the barcode prefix:
400638133393
Complete barcode: 4006381333938
```

```
Enter the barcode prefix:
4006381333d3
Invalid barcode prefix
```

```
Enter the barcode prefix:
123defg
Invalid barcode prefix
```

```
Enter the barcode prefix:
123
Invalid barcode prefix
```

Task 2: IBAN validation

Check digits are used for many codes. A more complex case is that of the IBAN (International Banking Account Number), which is "an internationally agreed upon system of identifying bank accounts across national borders to facilitate the communication and processing of cross border transactions with a reduced risk of transcription errors" (Wikipedia). Your task is to write a program to implement a variation on the IBAN validation process – that is, rather than computing the check digits as in Task 1, you will instead verify that a complete IBAN (including check digit) is valid.

The official process for validating an IBAN is given at https://en.wikipedia.org/wiki/International_Bank_Account_Number#Validating_the_IBAN. For this lab, you should follow the simplified and modified process below:

1. Read the IBAN from the user and remove all spaces (this code is provided).

2. Check that all characters are either numbers or capital letters; if not, you should print "Invalid IBAN" and return immediately; do not further process an IBAN that fails at this step.
3. Move the first four characters from the start of the code to the end. (You can assume that the string is at least four characters long.)
4. Replace each letter with two digits – for example, A -> '10', B -> '11', etc.
5. Interpret the result as a large integer.
6. If the number **mod 89** equals 1, the IBAN is valid, otherwise it is invalid. You should print "Valid IBAN" if it is valid, and "Invalid IBAN" if it is invalid.

Since the numbers will be extremely large, you will need to use the **BigInteger** class to implement the final check. Assuming that **ibanStr** is the string representing the result of the above processing steps, here is how to complete the final step of the validation process.

```
BigInteger bigInt = new BigInteger (ibanStr);
int modValue = bigInt.mod (BigInteger.valueOf (89)).intValue();
```

After the above lines, **modValue** will contain the result of computing the remainder mod 89, and you should just need to check if that is equal to 1 to complete the validation.

You should implement this algorithm by completing the code in the **main** method in the **IBANValidator.java** file.

Worked example

Here is a worked example based on one from Wikipedia (the colours help to follow the processes):

- IBAN: GB85 WEST 1234 5698 7654 32
- Remove spaces and rearrange: WEST12345698765432GB85
- Convert to integer: 3214282912345698765432161185
- Compute remainder mod 89: result is 1, so IBAN is valid

Sample output

Here are some examples of the code being run. Each of these segments shows a separate run, showing the content of the Terminal window, where user input is indicated in *italics*. Note: because we are using mod 89 (where the true IBAN algorithm uses mod 97), **you should not validate examples with online IBAN checkers**, because they will mark our examples as invalid.

```
Enter the IBAN:
GR45 0110 1250 0000 0001 2300 695
Valid IBAN

Enter the IBAN:
SA33 8000 0000 6080 1016 7519
Valid IBAN

Enter the IBAN:
SA03 8000 0000 6080 1016 7591
Invalid IBAN

Enter the IBAN:
SA03 8000 0000 6080 $$$$ 7519
Invalid IBAN
```

What to submit

Before the deadline, you should submit your final version of the two source files, **EANBarcodeCompleter.java** and **IBANValidator.java**.

In addition to the source files, you should also submit a text document (MS Word or plain text) including the following information:

- **For each task**, your file should include a **worked example** like the examples above, showing how your code processes a sample valid input. **Your example cannot use the same sample input as in the provided worked examples.** Include screenshots and/or typed steps if you want, but do not include code in the document.

Marking scheme

You will receive a mark out of 6 for this lab, 3 marks for each task, allocated as follows:

- 1 mark for submitting code for each task, whether correct or not.
- 1 mark for **completely correct** code for each task. "Completely correct" means the program runs without errors and produces correct output for all reasonable inputs, including invalid cases such as those shown in the examples above.
- 1 mark for the **worked example** for each task. Each example must be correct, must show your code processing a valid input, and must not be identical to the provided examples; otherwise, no mark will be awarded.

Optional tasks

If you have completed the lab tasks, here are some additional things you might want to try:

1. You can implement more complete validation on the IBAN input, for example making sure that the country code at the start is correct and that the length is correct for the country.
2. You could update the validation code so that the user is continuously prompted to enter input until they have entered something of the right format (e.g., correct length for barcode, only alphanumeric for IBAN).
3. You can also experiment with implementing other check digits, for example validating credit card numbers using the Luhn algorithm https://en.wikipedia.org/wiki/Luhn_algorithm.

Coming up

This lab was a stand-alone exercise to get you used to writing Java code in VSCode. From next week, and for the rest of the semester, we will develop and extend a game called “Library of Legends” … stay tuned for more information next week!