Java Programming 2 – Lab Sheet 2

This Lab Sheet contains material based on the material up to arrays and non-primitive types.

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 8 October 2020.

Aims and objectives

- Writing more complex Java methods
- Using arrays and advanced control structures
- Using built-in Java methods and classes (Character, String, BigInteger)

Submission material

Please refer to lab sheet #1 for instructions on launching and using JShell.

Suggestions of built-in Java methods that will be useful for these tasks are provided after the task descriptions.

Task 1: Validating bar codes

Your task is to implement a method to check whether an EAN-13 or EAN-8 barcode is valid. You can read more about EAN 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; an EAN-8 (8-digit) barcode is used on small packages where an EAN-13 would be too large.
>
> The final digit of every EAN-8 and 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 (or 7) digits using a sum of products as described below. An EAN is only valid if the check digit is correct.

So, to validate a bar code, you need to compute the check digit based on the first 12 (or 7) digits, and compare that digit to the final digit of the input bar code. The bar code is valid only if the computed check digit matches the last digit of the bar code.

Here is the process for computing a check digit, which is the same for EAN-13 and EAN-8:

1. The check digit is computed based on the first 12 (or 7) digits – you should ignore the last digit when doing this computation.
2. The calculation begins with **the rightmost digit**. Each digit, **from right to left**, is alternately multiplied by 3 or 1. These products are summed modulo 10 to give a value ranging from 0 to 9.
3. 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.

The following examples (based on the Wikipedia page) should make the process easier to follow. For EAN-13 barcode **4006381333931**, the check digit calculation is:

digits	4	0	0	6	3	8	1	3	3	3	9	3
weight	1	3	1	3	1	3	1	3	1	3	1	3
partial sum	4	0	0	18	3	24	1	9	3	9	9	9
checksum												89

89 modulo 10 is 9. If we subtract this from 10, then we compute a check digit of **1**. This matches the last digit, so the barcode is **valid**.

For EAN-8 barcode **73513536**, the check digit calculation is:

digits	7	3	5	1	3	5	3
weight	3	1	3	1	3	1	3
partial sum	21	3	15	1	9	5	9
checksum							63

63 mod 10 is 3. If we subtract this from 10, then we compute a check digit of **7.** This does not match the last digit, so this barcode is **invalid**.

Your task is to implement a barcode checker: given a string containing a barcode, your method should check whether it is valid, by first verifying that it has the correct length and format, and then computing the check digit and comparing it with the final digit in the input barcode.

The signature of your method should be as follows:

```
boolean checkBarcode (String barcode)
```

Concretely, the suggested procedure is as follows[1]:

- First, verify that the input string is a valid length – i.e., it should be either 8 or 13 characters long. If the length is invalid, you should **return false**.
- If the barcode length is valid, then you should compute the check digit as described above, using the first 7 (or 12) digits in the string.
    - If you encounter a character in the barcode that is not a digit, you should **return false**.
- If the computed check digit matches the final digit of the barcode, you should **return true**; otherwise, you should **return false**.

Hint: You will probably find it useful for each digit to keep track of whether it is even or odd.

[T] It doesn't matter if you don't follow this exact process, as long as your program works correctly.

Some possible results of running the method are as follows:

```
jshell> checkBarcode (“4006381333931”)
$5 ==> true

jshell> checkBarcode (“73513537”)
$6 ==> true

jshell> checkBarcode (“73543517”)
$7 ==> false

jshell> checkBarcode (“abcdefgh”)
$8 ==> false

jshell> computeChecksum (“123”)
$9 ==> false
```

Your code will be marked automatically, so be sure that your method signature and output are exactly as shown above.

You should ensure that your **checkBarcode** method is saved in a file **checkBarcode.java** that can be loaded into JShell.

Task 2: Generating check digits for an IBAN

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 system of identifying bank accounts across national borders to facilitate the communication and processing of cross border transactions with a reduced risk of transaction errors". Your task is to write a program to generate the checksum for an IBAN – that is, rather than verifying the input as in Task 1, you will instead generate a checksum for a given IBAN.

The process for generating the checksum for an IBAN is given at https://en.wikipedia.org/wiki/International_Bank_Account_Number#Generating_IBAN_check_digits – a summary is below.

1. Extract the first two characters of the IBAN – these indicate the country code (e.g., "GB" represents the UK). If the first two characters do not correspond to a valid country code, then **the IBAN is invalid**.
2. Next, check that the IBAN is an appropriate length for the given country (with spaces removed). The following table indicates the countries that we will consider for this lab, and the valid IBAN length for each. If the IBAN length is not appropriate for the given country, then **it is invalid**.
    - GB: 22 characters
    - GR: 27 characters
    - SA: 24 characters
    - CH: 21 characters
    - TR: 26 characters
3. If the IBAN is invalid, stop here. If it is in a valid form, begin processing it, as follows.
4. Move the first four characters from the start of the IBAN to the end, and replace the final two characters with "00".
5. Create a new string of characters as follows:
    - Copy each number across directly
    - Replace each letter in the string with two digits, thereby expanding the string, with A = 10, B = 11, … Z = 35.
    - If you encounter any character that is not a number or a letter, then **the IBAN is invalid**.
6. Interpret the resulting string as a (very large!) integer and compute the remainder of that number when dividing by 97.
7. Subtract the remainder from 98 and use the result for the checksum.

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

- IBAN: GB12 WEST 1234 5698 7654 32
- Rearrange and convert to 00 if necessary: W E S T 12345698765432 G B 00
- Convert to integer: 3214282912345698765432161100
- Compute remainder mod 97: result is 16
- Compute 98 – 16: checksum is **82**

Your task is to implement a method with the following signature:

int computeChecksum (String iban)

The method should proceed as follows:

- First, remove all spaces from the input string and convert it to uppercase – you can do this as follows:
  iban = iban.replaceAll(" ", "").toUppercase();
- If the country code is not valid (i.e., is not one of the countries listed above), you should print "Unknown country code: (country)" and **return -1**. For example:
  - `computeChecksum("ZZ 1234 5678");`
    `Unknown country code: ZZ`
    `$10 ==> -1`
- If the country code is valid, but the IBAN is not the correct length for that country, you should print "Invalid IBAN length: (length)" and **return -1**. For example:
  - `computeChecksum("GB93 0076 2011 6238 5295 7");`
    `Invalid IBAN length: 21`
    `$11 ==> -1`
- If the country code and length are valid, you should process the string as described above: moving the first four characters to the end, changing the last two characters to "00", and then creating a new string where all letters are replaced by an appropriate number ('A' by 10, 'B' by 11, etc.).
  - If you discover during this process that the IBAN contains an invalid character (i.e., a character that is not a letter or a number), you should print "Invalid character in IBAN: (character)" and **return -1**. (Note that you should print this for the **first** invalid character that you encounter if there are several.) For example:
    - `computeChecksum("CH93 0!76 2011 6*38 5295 7");`
      `Invalid character in IBAN: !`
      `$12 ==> -1`
- If all of the above checks pass, then you should compute the remainder of the new (very large!) number mod 97. As the numbers involved in IBANs are much too big to represent in a Java **int**, or even a **long**, you will need to use the **BigInteger** class for this. Assuming that you have a **String** variable called **digits** containing the result of the preceding steps, here is how to get the remainder and store it in a new variable called **mod**:
  int mod = new BigInteger(digits).mod(BigInteger.valueOf(97)).intValue();
- Finally, you can use the value of **mod** to compute the checksum by subtracting it from 98, and return that as the result of your method.

Here are some sample inputs to use in your testing (adapted from the Wikipedia article):

- GR12 0110 1250 0000 0001 2300 695 (checksum should be 16)
- GB34 NWBK 6016 1331 9268 19 (checksum should be 29)
- SA56 8000 0000 6080 1016 7519 (checksum should be 3)
- CH78 0076 2011 6238 5295 7 (checksum should be 93)
- TR90 0006 1005 1978 6457 8413 26 (checksum should be 33)

You should ensure that your **computeChecksum** method is saved in a file **computeChecksum.java** that can be loaded into JShell.

Hints and tips

The following built-in Java tools are potentially useful for both tasks.

1. You might need to access *substrings* of a String – that is, parts of the string between a starting and an ending index. If **str** is a String, then you can use the **substring()** method to do this as follows (it can do more, but these are the relevant uses):
    - **str.substring(0, 2)** returns a String consisting of the first two characters of str
    - **str.substring(4)** returns a String consisting of the characters after the fourth character of str
2. You can convert a string into a character array using the **toCharArray()** method. If **str** is a String, then this is how to use it:
    - **char[] chars = str.toCharArray();**
3. If **c** is a **char**, then you can do the following things:
    - **Character.isDigit(c)** returns true if **c** is a digit (0-9) and false if not
    - **Character.isLetter(c)** returns true if **c** is a letter (a-z, A-Z) and false if not
    - **Character.getNumericValue(c)** returns an **int** corresponding to the numeric value of the character **c**. For example:
        i. **Character.getNumericValue('5')** returns the int 5
        ii. **Character.getNumericValue('A')** returns the int 10
4. To help in debugging your methods, it might be useful to use System.out.println() to display the intermediate results. **DO NOT FORGET TO REMOVE ANY SUCH STATEMENTS BEFORE SUBMITTING THE FINAL VERSION.**
5. Please test your code with all of the examples shown above. We will be expecting the exact output shown, including spelling and punctuation, so you should double check before submitting.
6. Don't forget that **switch** can also be used on Strings – this might be useful for validating country codes and lengths.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JP2 moodle site. Click on Laboratory 2 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code and drag only the following files into the drag-and-drop submission area:

- **checkBarcode.java**
- **computeChecksum.java**

Then click the blue save changes button. Check the two .java files are uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via moodle.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" (*). We will automatically execute your submitted code and check its output; we will also look at the code before choosing a final mark.

Example scores might be:

5*: you complete both tasks correctly with no bugs

4*: you complete one of the two tasks correctly and have made an attempt at the other

3*: you completed only one of the tasks, or have made a reasonable attempt at both

2*: you have made some attempt at both tasks

1*: minimal effort

For this assignment, we will not deduct for inappropriate code formatting or commenting – however, the tutors may comment on such stylistic issues and will deduct for them in the future.

Possible extensions

If you have completed the lab assignment and want to try some extra tasks, here are some ideas for things to try. Please **be sure that you do not submit code for any of these tasks** – the tutors only want your solutions to the assessed tasks above.

1. Write a method to generate check digits for barcodes instead of validating them
2. Write a method to validate IBANs instead of generating the checksum (see https://en.wikipedia.org/wiki/International_Bank_Account_Number#Validating_the_IBAN for details of the process)
3. Try validating or generating other sorts of check digits – for example, here are the details of how to validate a credit card number: https://en.wikipedia.org/wiki/Payment_card_number