Laboratory Sheet 2

This Lab Sheet contains material primarily based on Lectures 3 – 4.

The deadline for Moodle submission of the lab exercise is 17:00 on Thursday 6 October.

You may submit work that is incorrect or incomplete. In order to stretch the stronger members of the class, some of the exercises are quite challenging so don't worry if you can't complete all of them. You should spend around 3 hours per week on programming exercises.

Default login details: Username = matric + 1st initial of family name, password = last 8 digits of library barcode.

Beginners – start at 'Set Up' Section on page 1.

Experts – skip to 'Submission material' Section on page 2.

Aims and objectives

- Processing Strings and characters.
- Converting between data types
- Using advanced control structures

Set Up

1. Download Laboratory2.zip from Moodle and launch Eclipse.
2. In Eclipse, select File → Import ... to launch the import wizard – this will be used to import the starter projects into your workspace. Then select General → Existing Projects into Workspace and click Next.
3. Choose Select archive file and then browse to the location where you downloaded Laboratory2.zip. Select Laboratory2.zip and click Open.
4. You should now see a set of two projects in the Projects window: Submission2_1 and Submission2_2. Ensure that the checkboxes beside both of these are selected (e.g., by pressing Select All if they are not), and then press Finish.
5. In the *Package Explorer view* (on the left of the Eclipse window), you should now see icons representing the two new projects (in addition to any projects that were there before).

Please refer to the lab sheet from Laboratory 1 for additional details on how to open and run the code in the two new projects.

Submission material

Preparatory work for these programming exercises, prior to your scheduled lab session, is expected and essential to enable you to submit satisfactory solutions.

Unit Tests: I have supplied some simple test cases to check that your code is working properly. When you create each of the projects, you should see *two* source code files – one is the skeleton Java source code for you to fill in the details. The other file contains the Unit tests to check your programs are working properly.

To execute the program:
right click on EanChecksum or IbanChecker -> run as Java application
right click on EanChecksumTest or IbanCheckerTest -> run as JUnit test

Submission 1

Your task is to implement a program to compute the *check digit* for an EAN-13 or EAN-8 barcode. 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.

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

1. 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.
2. 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 (from the Wikipedia page) should make the process easier to follow. For EAN-13 barcode **400638133393**, 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**

For EAN-8 barcode **7351353**, 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.**

Your task is to implement the above process: given the initial 12 (or 7) digits of a barcode, your program should compute and output the check digit. Your program should also verify that the input is valid and output appropriate error messages if it is not.

Concretely program should operate as follows:

- Input to the program should come from the keyboard. This is already set up in the `main` method, which you will not need to change.
- After the EAN has been read, you should next check that it is a valid length – i.e., the input string should be either 7 or 12 characters long. You can access the string length by calling `barcode.length()`. If the length is invalid, you should print the message "Invalid barcode length" and exit the main method by writing `return;`[1]
- If the barcode length is valid, then you should implement the above calculation. You can access the characters in the string using `barcode.charAt(i)`
- If you encounter a character in the barcode that is not a digit (i.e., it is not in the range '0' to '9'), you should print the message "Invalid barcode format" and exit using `return;`
- If the barcode is valid, you should output the computed check digit in a line that looks like this: "Checksum digit: *n*" (where *n* is the digit you computed)

Here are sample runs (user input is indicated with italics):

```
Enter the barcode number:
400638133393
Checksum digit: 1

Enter the barcode number:
abcdefg
Invalid barcode format

Enter the barcode number:
123
Invalid barcode length
```

I have provided skeleton starter code: in the *Package Explorer view* (on the left of the Eclipse window), open the Submission2_1 project and find the single source code file for this program, i.e. EanChecker.java. Expanding the default package icon will reveal this file. Then double click the file name to open the file in the central *Editor view*. You should write the code implementing the above specification where indicated inside the main method. Note that the code for prompting the user for the barcode and reading the result is already included for you and should not be changed.

Hint 1: You will probably find it useful to keep track for each digit whether it is even or odd.

Hint 2: The suggested method of accessing the characters of the string (`barcode.charAt`) returns a `char`. You will need to convert from a `char` (e.g., '5') to an `int` (e.g., 5): one way to do this is to subtract the char '0' from the value (don't forget that char is a numeric type). Other possibilities also exist.

---

[1] Normally in this situation a Java programmer would probably write `System.exit(0);` or similar rather than `return;` But unfortunately that doesn't interact well with the JUnit test cases so I'm asking you to return instead of exiting.

Hint 3: The provided test cases give a variety of inputs that should produce all of the output above. If a test case is failing, you should check whether you are producing output that exactly matches the examples above, including spelling and punctuation.

Submission 2 (more challenging)

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 validate an IBAN – that is, rather than computing the check digits as in Submission 1, you will instead verify that a complete IBAN (including check digit) is valid.

The process for validating an IBAN is given at https://en.wikipedia.org/wiki/International_Bank_Account_Number#Validating_the_IBAN; the 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. Now, begin processing the IBAN. First, move the first four characters from the start of the IBAN to the end.
4. Next, 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 = 26.
    - If you encounter any character that is not a number or a letter, then **the IBAN is invalid**.
5. Finally, interpret the resulting string as a (very large!) decimal integer and compute the remainder of that number when dividing by 97. If the result is 1, then **the IBAN is valid**; otherwise, **the IBAN is invalid**.

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

- IBAN: GB82 WEST 1234 5698 7654 32
- Rearrange: W E S T12345698765432 G B82
- Convert to integer: 3214282912345698765432161182
- Compute remainder mod 97: result is 1, so IBAN is valid

The starter code in IbanChecker.java is the starting point for your implementation of the IBAN validity checker. This code reads an IBAN from the user, removes all spaces, converts all letters to upper case, and stores the result in the String variable `iban`. You should implement all of the above checks on the given IBAN and output an appropriate message, as follows:

- If the country code is not valid, you should print "Invalid country code: (country)" and exit with a `return;` statement. For example:
  - ```
    Enter the IBAN:
    ZZ 1234 5678
    Invalid country code: ZZ
    ```
- 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 exit with a `return;` statement. For example:
  - ```
    Enter the IBAN:
    GB93 0076 2011 6238 5295 7
    Invalid IBAN length: 21
    ```
- If the country code and length are valid, but 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 exit with a `return;` statement. (Note that you should print this for the **first** invalid character that you encounter if there are several.) For example:
  - ```
    Enter the IBAN:
    CH93 0!76 2011 6*38 5295 7
    Invalid character in IBAN: !
    ```
- If all of the above checks pass, then you should compute the remainder of the new (large!) number mod 97. As the numbers involved in IBANs are much too big to represent in a Java `int`, or even a `long`, I have provided an implementation of mod97 that processes a number represented as a string and returns the remainder mod 97. You can call it in your code like this (assuming you have a String variable called `digits` containing the result of the preceding steps):
  - ```
    int mod = mod97(digits);
    ```
- Finally, you can use the value of `mod` to determine whether the IBAN is valid or invalid. If it is equal to 1, then it is valid; otherwise, it is invalid. Here are two sample outputs for the two cases:
  - ```
    Enter the IBAN:
    TR33 0006 1005 1978 6457 8413 26
    IBAN is valid
    ```
  - ```
    Enter the IBAN:
    TR33 0006 1005 1978 6457 8413 62
    IBAN is invalid
    ```

Here are some sample valid IBANs to use in your testing (taken from the Wikipedia article); these are also the IBANs that are used in the JUnit test cases:

- GR16 0110 1250 0000 0001 2300 695
- GB29 NWBK 6016 1331 9268 19
- SA03 8000 0000 6080 1016 7519
- CH93 0076 2011 6238 5295 7
- TR33 0006 1005 1978 6457 8413 26

Hints and tips

Hint 1: in addition to using the `length()` and `charAt(i)` method on Strings as in Part 1, this program will also require you to access *substrings* of a String – that is, parts of the string between a starting and an ending index. 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`

Hint 2: to help in debugging your program, it might be useful to use `System.out.println()` to display the intermediate strings built during the IBAN validation process.

Hint 3: the test cases can be used to verify that your code works properly. The test cases are expecting the exact output shown in the examples above, including spelling and punctuation, so you should double check that if things are failing.

Hint 4: don't forget that `switch` can also be used on Strings – this might be useful for validating country codes and lengths.

How to submit

Please 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 JOOSE2 moodle site. Click on Laboratory 2 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably H:\workspace\Submission2_1\ -- and drag only the *single* Java file EanChecksum.java into the drag-n-drop area on the moodle submission page. Then do the same for file IbanChecker.java in H:\workspace\Submission2_2\. **Your markers only want to read your java files, not your class files.** 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 A1-H. Example scores might be:

A1: you complete both submissions correctly and pass all the unit tests
B1: you complete one of the parts correctly and pass all of its tests
C1: you attempt either or both methods and show that you are trying to implement a sensible algorithm, even if it does not work properly.

For this assignment (and all future ones), we will also be considering code style – comments, formatting, variable names, etc – in addition to correctness. You will be deducted some marks (e.g., from an A1 to and A3) if your code style is not appropriate. Please check with the tutors and demonstrators during your lab session if you are uncertain about style.