Exercises module 22 – Generic collections

22.1 – Dictionary – Decimal to Hex conversion

1. We need to make a class that converts a decimal number (0-255) to its corresponding hexadecimal value.

   Create a new project and in your main method add the following code:

```
HexConverter hexconv = new HexConverter();

System.out.println(hexconv.convertToHex(255));   //0xFF
System.out.println(hexconv.convertToHex(240));   //0xF0
System.out.println(hexconv.convertToHex(128));   //0x80
System.out.println(hexconv.convertToHex(15));    //0x0F
System.out.println(hexconv.convertToHex(1));     //0x01
System.out.println(hexconv.convertToHex(0));     //0x00
```

2. Create a new class **HexConverter** and add a private **HashMap** field with an integer as the **key** type and a Character as the **value** type. Name it **decToHexMap**.

3. Add a **default** constructor to the class that populates the HashMap with the key 0-15 (16 entries) and their corresponding character. We will use this to map from decimal to hexadecimal.

4. Now we need to implement the **convertToHex** method. As a starting point it can be wise to create a private method named **lookupHexValue** that takes an integer between 0-15 and returns the corresponding hex character.

5. Implement the rest of the logic in the **convertToHex** method using the lookup helper method. Start with just supporting input values **0-15**.

   Advanced - Supporting larger numbers
   if you have time, try to also add support for numbers between 0-255

   A useful term to be aware of is Nibble, read more about it here:
   https://en.wikipedia.org/wiki/Nibble

   To split the incoming decimal value (0-255) into its high and low nibble parts:

```
int highNibble = (input >> 4) & 15;
int lowNibble = input & 15;
```

   (Where & means a **bitwise and** operation and >> means a **binary shift right**. If you have time try to understand the logic behind this)

6. Add the remaining necessary code to make the method work. Basically converting the two integers above into their characters and returning them back to the caller.

   Bonus feature
   If you really have time, you could try to add support for numbers between 0-65535 (0xFFFF)

22.2 – Dictionary

This time we are going to explore the world of dictionaries and string manipulations.

Given an input string, like: `Hello world!`

1. Count the number of times each unique character appears.
   For both cases print out the result similar to the following chart:

```
Counting the characters in the string 'Hello World!'

  *
! *
r *
d *
e *
W *
H *
l ***
o **

The most common character is "l" that appeared 3 times.
```

   (Where each star represents one occurrence of that character. The order of the output does not matter.)

   Hints:
   1. Before you start coding, explore how you would solve this task using only pen and paper
   2. Separate the calculation from the printing out the result. The **Separation of Concerns** principles is a good principle here to simplify the code.
   https://en.wikipedia.org/wiki/Separation_of_concerns

2. Print out the most common character like “**The most common character is ‘l’ that appeared 3 times.**”

3. Read more about Histograms here: https://sv.wikipedia.org/wiki/Histogram

22.3 – Stacks

1. Write a program that reverses the List of integers in the code below using a stack.

```
List<Integer> nums = new ArrayList<>();
nums.add(1);
nums.add(2);
nums.add(3);
nums.add(4);

//Reverse the nums List using a stack

for (Integer num :nums)
    System.out.println(num);
```

The output should be 4,3,2,1.

22.4 – Queues – Producer/Consumer

1. Queues are useful to separate a consumer from the producer of jobs. In the code below, we have:
   a. a method named "Producer" that should add data to the queue
   b. a method named "Consumer" that removes items from the queue
   c. We print out the queue length

```
    public static void main(String[] args) {
// write your code here

        Scanner scanner   = new Scanner(System.in);

        Queue<Integer> queue = new LinkedList<>();

        while(true)
        {
            Producer(queue);
            Consumer(queue);

            System.out.println("Queue length: " + queue.size());

            scanner.nextLine();
        }
    }
```

2. Implement the producer and consumer methods so that:
   a. The **producer** method randomly adds 0-5 items to the queue each time it is called. The items could just be a number, the exact number is not that interesting right now.
   b. The **Consumer** method should remove 0-5 items each time it is called.

3. Run the application and watch the queue size grow and shrink over time, like:

```
Queue length: 7
Queue length: 8
Queue length: 8
Queue length: 8
Queue length: 6
Queue length: 6
Queue length: 7
Queue length: 8
Queue length: 8
Queue length: 6
Queue length: 10
Queue length: 10
```

22.5 – Hash values

1. Dictionaries and other data structures often depends on **hash functions** and **hash values** internally. Hash functions is also an important function in computer security.

   For example what does this code do?

```java
String Str = "Academy java";
System.out.println("Hashcode=" + Str.hashCode() );
```

2. Read about hash functions here https://en.wikipedia.org/wiki/Hash_function and on the web and try to understand what a hash function and value is and what the code above does. https://en.wikipedia.org/wiki/Hash_function
3. Also do read and understand what a checksum is: https://en.wikipedia.org/wiki/Checksum
4. Visit http://demo.edument.se/Encryption and try the online hash generator. If you search for "summer" and then take the hash value and search for it on the web, can you find the original text just by searching for the hash value?

Things to explore further:

- Producer consumer pattern
  https://dzone.com/articles/producer-consumer-pattern
- Producer / Consumer problem
  https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem
- Linked lists
  https://en.wikipedia.org/wiki/Linked_list
- Checksums
- Hash values and functions