Exercise using a Map (files found in the **MapEx** project):

For this exercise you will store **Product** objects in a Map. Every Product has a Product Code (**ProductCode**). We'll assume that ProductCodes are unique -- there will never be two products with the same code. If we are ever looking at two Product objects and they have the same ProductCode, then they are the same Product.

A **ProductCode** consists of 3 pieces of information:

1. The product's **country of origin**. This is just a character such as '**U**' for USA, '**C**' for China, '**N**' for the Netherlands, or '**I**' for India.
2. The **department code**. This is the department that the product is sold in and is represented by an integer. For example **101** might be the electronics department, while **212** might be the gardening department.
3. The **manufacturer's code**. This is the code used by the product's manufacturer

The three pieces of information are combined to generate the Product code. For example, a product whose manufacturer is in the USA: '**U**', sold in our electronics department **101**, and having the manufacturer code of **A584321** would have the following product code:
U-101-A584321

When running the program, provide **3 command line arguments** -- each piece of one ProductCode. For example, if we wanted to lookup the above product, we would run the program as follows:

```
java MapExample U  101  5C459
```

To save everyone time I have written a class **ProductFileReader** which simulates reading information from a file. It doesn't actually read the data from a file, the data is hardcoded. But we could easily modify the ProductFileReader class later to read from a file without having to change any of the other classes that use it. This can be done thanks to the idea of **information hiding.**

For this exercise:

1. Look over the class **MapExample**. This class has a main() function that reads data from a file (using **ProductFileReader**), stores that data in a map, builds a **ProductCode** from the command line arguments and then does a lookup to see if that Product is in the Map. Finally the contents of the Map are printed.
2. Modify the function **readProdMap()** (in class MapExample) which reads Products from a file, one by one and adds them to the Map prodMap.
3. Add any necessary functions to **Product** or **ProductCode** for your Map to work correctly
4. Modify the function **printProdMap()** (in class MapExample) which prints the contents of the Map **prodMap**.
5. Modify the main() function so that if the **ProductCode** specified on the command line is found, it **prints** the Product. If the ProductCode is not in the data structure, **print** a message saying the Product code is invalid.

Here is a sample run of the program:

java MapExample C 101 654-1A

```
*** Result of Product Lookup ***
Product: microwave Code: C-101-654-1A
        Mfg Price: $69.99 Sale Price: $99.90

The Product Map:
Product: toaster Code: U-101-A584321
        Mfg Price: $25.50 Sale Price: $59.90
Product: microwave Code: C-101-654-1A
        Mfg Price: $69.99 Sale Price: $99.90
Product: tulip bulbs 12-pack Code: N-1120-C-102A-1111-26
        Mfg Price: $2.00 Sale Price: $9.50
```

… (Additional output here)