Practical exercise 2

Expend our Employee Expenses Management Application project as follows:

1. Create the following custom exceptions in a separate package. The exceptions do not need to take encompass any kind of message.

   `InvalidEmployeeIdException`

   `NameTooShortException`

2. Create a method in the EmployeeUtilities class called `validateEmployeeId`. This method should take a single parameter of type String, and return an Integer. The implementation should be:
   - if the string can be cast to an integer, then do the cast and return it
   - if the string cannot be cast to an integer, throw an InvalidEmployeeIdException.

   Hint: convert a string into an integer with the Integer.valueOf() method. This method might throw a NumberFormatException

3. Create a void method in the same class called `validateEmployeeName`. This method should take two string parameters to represent the firstName and surname. If the combined length of both fields is fewer than 6 characters then throw a NameTooShortException, otherwise do nothing.

4. Move the RegisterNewEmployee class into a new package called `ui` (user interface)

5. In the RegisterNewEmployee class, collect all of the fields to register the new employee (except for the expense claims). Implement the following:

   - The ID of the new employee should be read in from the console as a String, and use the validateEmployeeId method to convert it to an integer. Catch the exception, and repeat this step until a valid ID is entered.
   - Where the user can input the firstName and surname, use the validateEmployeeName method to check the input is sufficient – if it isn't then repeat these steps until it is.
   - where the user can input a department, this will currently crash if you do not enter a valid department name – catch this exception and repeat the step until a valid name has been entered.

6. Make the method in the RegisterNewEmployee class a non runnable method called `registerNewEmployee`, which will return the employee that has been created. Remove the line that prints out the employee at the end of the method, and refactor the class name to `UIFunctions`.

7. Edit the Employees class and add the following new methods:
   - Check to see if an employee exists in the array – this method should require the ID of the employee as a parameter

- Add an expense claim to an employee – this method should require an expense claim as the parameter, then find the relevant employee and add this claim to the employee's claims array. Throw an exception of your choosing if the employee isn't found. Don't check if the claims array is full – just expect there to be sufficient space for the new claim.

8. Edit the ExpenseClaim class and change the data type of the dateOfClaim field to a LocalDate.

9. Edit the Employee class and ensure that the claims array is initialised as a new array of size 10 in the constructors.

10. Create a method in the UIFunctions class called `registerNewExpenseClaim`. This method should return a new expense claim. As part of the implementation:
    - Don't validate the id entered for the expense claim ID and employee ID is a number, or that the amount is a double – just assume it is;
    - Set the dateOfClaim to be the current date.

11. Create a new runnable class called `ExpenseManagementSystem` which will instantiate an instance of the Employees object, with an array of size 10 and then provide the following functionality:
    - a menu with the options to register a new employee, register a new expense claim, print a list of all employees to the screen, or exit the application.
    - For each of the registration options, use the methods just created, and then call the relevant methods in the Employees class
    - implement the options using the existing methods, and when each option has completed return to the menu.