Exercise 6

Exception Handling Functional Programming using Scala

- Last Week Recap [**Pattern Matching** (*It allows you to match values against patterns and execute code based on the match),* **Type Parameterization** (*It allows creating generic code that can work with different types of data),* **Variance** (*A mechanism for defining subtyping relationships between generic types. It determines whether a type parameter can be replaced by a subtype or a supertype):* **+: Covariant, -: Contravariant and =: Invariant annotations**]
- In Scala, exception handling is similar to other programming languages such as Java. You can handle exceptions using the **try-catch-finally** block, which allows you to catch and handle any exceptions that may occur during the execution of your code.
    - The **try** block contains the code that may throw an exception
    - If an exception occurs, the **catch** block will handle the exception by matching it to a specific exception type
    - The **finally** block is optional, and contains code that will always be executed whether an exception occurs or not
- **Drawback**s:
    - The syntax results in scoping issues leading to unnecessarily complicated constructs as well as the need to nest try-catch block within either the try, catch or finally part of the top-level construct.
    - Multi-threaded code can be difficult to deal with using the try-catch-finally construct. For example, how should you react to an exception which occurs in a separate thread but which impacts the data being accessed? It assumes that the exception is handled in the current thread.
    - Try-catch-finally approach is not particularly functional and is more procedural in nature.
- **Alternative Solutions**:
    - ***Option*** Data Type in Scala
      Option has two cases: it can be defined, in which case it will be a **Some**, or it can be undefined, and in which case it will be **None**.
    - ***Either*** Data Type in Scala
      Either has only two cases **Left** & **Right**, just like ***Option***. The essential difference is that both cases carry a value. The Either data type represents, in a very general way, values that can be one of two things.
    - Using any of ***Option*** or ***Either*** types for error handling makes the code more readable and avoid the need for using try-catch blocks.

Classwork

In this lab, you will be exploring the fundamentals of conventional exception handling and alternatives i.e. Option & Either data types available in Scala for exception handling in a more functional programming manner.

Homework

NOTE: You should use immutable data types and avoid using any mutable variables or loops. Also do not use conventional exception handling approach for doing the tasks defined below.

Task 1

Create a function **lookingForPrimes** that takes a list of **integers** and returns a list of **Either** type values. If the integer is prime, return **Right** with the prime integer. If the integer is not prime, return **Left** with an error message. Use **map** to apply this function to a list of integers and then use **foreach** to print the results or error messages.

Task 2

Create a **case class** for a student that contains a name, age, and grade (an **Option[Int**]). Create a list of students with some missing grade values. Use **flatMap** to extract the grade values from each student, resulting in a list of **Int** values wrapped in **Option**. Then use **filter** to remove any **None** values and then to find the average grade of the remaining students using built-in function or recursion. (Important: No Loops)

Task 3

Create a function that takes two **integers** and returns an **Either** value. If the second integer is zero, return **Left** with an error message. Otherwise, return **Right** with the result of dividing the first integer by the second integer. Create **a list of tuples** containing pairs of integers, and use a **higher order function** to apply this function to each pair, resulting in a list of **Either** values. Then use **partition** to separate the Right values from the **Left** values, and use **foldLeft** to find the sum of the **Right** values.

Task 4

Create a function that takes a list of **Strings** and returns an **Option** value. If any of the Strings in the list contains the word "error", return **None**. Otherwise, return **Some** with the concatenated string of all strings in the list. Use a **higher order function** to apply this function to a list of **Strings** and then use **match** to handle the **Option** value. If **Some**, print the concatenated string. If **None**, print an error message.

Deliverable

Deliverable: Submit <u>a single scala code file with "*.scala*" extension</u>, and write your Name and Student ID in the code as a comment. The whole deliverable must be well commented and supported with descriptions where required.
Deadline: 28.04.2023 12:00 am [Before Next Friday Session]
Submission: (session respective) Return box on Moodle.
Estimated workload: <= 2 hours
Warning: This is individual work. Strict actions will be taken for plagiarism!

Deliverables for Exercise 6:

1. Implementation of the homework part.

Note:
Make sure to follow the Scala style guide and use appropriate naming conventions for variables, functions, and classes. Your code should be well-organized, well-documented, and easy to read.