Scala Project

#tdt4165/project

Task 1.1

```
class TransactionQueue {
  private val queue = mutable.Queue[Transaction]()

  // Remove and return the first element from the queue
  def pop: Transaction = this.synchronized {
    queue.dequeue()
  }

  // Return whether the queue is empty
  def isEmpty: Boolean = this.synchronized {
    queue.isEmpty
  }

  // Add new element to the back of the queue
  def push(t: Transaction): Unit = this.synchronized {
    queue += t
  }

  // Return the first element from the queue without removing it
  def peek: Transaction = this.synchronized {
    queue.front
  }

  // Return an iterator to allow you to iterate over the queue
  def iterator: Iterator[Transaction] = this.synchronized {
    queue.iterator
  }
}
```

Task 1.2/1.3

```
def withdraw(amount: Double): Either[Unit, String] = {
  this.synchronized {
    if (amount < 0 || amount > balance.amount) return Right("Insufficient
funds")
    Left(balance.amount -= amount)
  }
}

def deposit(amount: Double): Either[Unit, String] = {
  this.synchronized {
    if (amount < 0) return Right("Amount cant be negative!")
    Left(balance.amount += amount)
  }
}

def getBalanceAmount: Double = this.synchronized {
  balance.amount
}
```

Task 2

```
def addTransactionToQueue(from: Account, to: Account, amount: Double): Unit = {
  transactionsQueue push new Transaction(
    transactionsQueue, processedTransactions, from, to, amount, allowedAttempts
  )

  Main.thread(processTransactions)
}
```

```
private def processTransactions: Unit = {
  val transaction: Transaction = transactionsQueue.pop
  transaction.run()

  if (transaction.status == TransactionStatus.PENDING) {
    transactionsQueue push transaction
    processTransactions
  }
  else {
    processedTransactions push transaction
  }
}
```

Task 3

```
override def run: Unit = {

  def doTransaction() = {
    val withdraw = this.from.withdraw(amount)

    withdraw match {
      case Left(_) => {
        to.deposit(amount)
        status = TransactionStatus.SUCCESS
      }

      case Right(_)
      => {
        attempt += 1
        if (attempt >= allowedAttempts) {
          status = TransactionStatus.FAILED
        }
      }
```

```
    }
  }

  if (status == TransactionStatus.PENDING) {
    this.synchronized {
      doTransaction
      Thread.sleep(50)
    }
  }
}
```