LU11.L02 - Abstrakte Klasse für Taschenrechner

main.py

```
""" main routine for the calculator program """
from calculator import Calculator
from exceptions import OperationException, NumberFormatException
from tokenizer import Tokenizer


def main():
    """
    Creates the Tokenizer and Calculator objects
    Runs the calculator
    - Read the calculation from the user
    - Determine the operation
    - Perform the calculation
    All exceptions are caught and printed
    """

    calc = Calculator(Tokenizer())
    try:
        calc.read_input()
        calc.create_concrete_op()
        calc.calculate()
    except OperationException as op_ex:
        print(op_ex)
    except NumberFormatException as nf_ex:
        print(nf_ex)
    except ZeroDivisionError:
        print('ERROR: Division mit 0!')


if __name__ == '__main__':
    main()
```

exceptions.py

```
""" Provides custom exceptions for the calculator. """
class OperationException(Exception):
    """
    This exception is raised when no valid operation sign is recognized
    during the split of the input.
    """
    def __init__(self):
        super().__init__('ERROR: ungültiges Operationszeichen
```

```
eingegeben!')


class NumberFormatException(Exception):
    """
    This exception is raised when an invalid number format is detected.
    """
    def __init__(self, value):
        super().__init__(f'ERROR:  {value}  ist ein ungültiger
Zahlenwert')
```

math_operations.py

```
""" Provides an abstract class for mathematical operations. """
from abc import abstractmethod, ABC


class MathOp(ABC):
    """
    An abstract class representing any (binary) mathematical operation.
    The method execute_op is abstract and must be overridden by
concrete classes.
    The method result returns the result of the executed operation.
    Note: unary operations like the factorial (!) cannot be calculated.
    """

    def __init__(self):
        """
        Initializes the result of the operation.
        """
        self._result = 0.0

    @abstractmethod
    def execute_op(self, val1, val2):
        """
        Defines the interface for the calculation of a binary operation
(operation with 2 values).
        The method receives two values as parameters and then performs
the appropriate operation.
        The concrete operation is determined in the derived class.
        :param val1: first numerical value
        :param val2: second numerical value
        """
        pass

    @property
    def result(self):
        """
        Returns the result of the mathematical operation.
```

```
        :return: result of the operation
        """
        return self._result
```

calculator.py

```
""" Provides the Calculator class. """
from adder import Adder
from divider import Divider
from multiplier import Multiplier
from reader import Reader
from subtractor import Subtractor


class Calculator:
    """
    Provides a class for the calculator.
    The calculator reads the user input, splits it into its components,
    creates the concrete operation and executes it.
    """

    def __init__(self, tokenizer_object):
        """
        Creates a Calculator object.
        Initializes the Reader object and assigns the Tokenizer.
        :param tokenizer_object: Tokenizer object
        """
        self._math_op = None
        self._my_reader = Reader()
        self._tokenizer = tokenizer_object

    @property
    def math_op(self):
        """
        Returns the reference of the currently created MathOp object.
        :return: reference of the MathOp object
        """
        return self._math_op

    def read_input(self):
        """
        Reads a value from the keyboard and passes the string to the
splitter.
        The exceptions that occur are only processed in the main
program!
        """
        self._my_reader.screen_info()
        value = self._my_reader.read()
        self._tokenizer.split(value)
```

```
    def create_concrete_op(self):
        """
        Factory method for creating the concrete operation.
        The concrete operation is determined by the operation sign.
        If the operation sign is not recognized, the reference is set
to None.
        Note: This case should never occur, as the Tokenizer otherwise
throws an exception.
        But for safety reasons, it should be implemented this way.
        """
        if self._tokenizer.operation == '+':
            self._math_op = Adder()
        elif self._tokenizer.operation == '-':
            self._math_op = Subtractor()
        elif self._tokenizer.operation == '*':
            self._math_op = Multiplier()
        elif self._tokenizer.operation == '/':
            self._math_op = Divider()
        else:
            self._math_op = None

    def calculate(self):
        """
        Executes the operation created in create_concrete_op.
        Note: No exceptions are processed.
        """
        if self._math_op is not None:
            self._math_op.execute_op(self._tokenizer.value1,
self._tokenizer.value2)
            print(f'Ergebnis: {self._math_op.result}')
```

adder.py

```
""" Provides the Adder class. """
from math_operations import MathOp


class Adder(MathOp):
    """
    Adds two numbers.
    """
    def execute_op(self, val1, val2):
        """
        Executes the operation val1 + val2.
        The result can be read via the getter method of result.
        :param val1: first numerical value
        :param val2: second numerical value
        """
```

```
        self._result = val1 + val2
```

M320-LU11