**Задание на лабораторную работу №5**

**Тема**: Обработка и генерация исключений. Пользовательские исключения.

* За основу взять лабораторную работу №4.
* Продемонстрировать возможности языка по генерации и обработке исключений, блоки: try/except, finally и raise.
* Применить assert для проверки предположения о значениях произвольных данных в произвольном месте программы.
* Создать и задействовать свое собственное исключение.
* Проверить работу программы.

Ход работа:

from abc import ABCMeta, abstractmethod  
  
# Определение класса R для использования в композиции  
class R:  
 def \_\_init\_\_(self):  
 print("r")  
  
# Определение класса Re, использующего композицию с классом R  
class Re:  
 def \_\_init\_\_(self):  
 self.\_rer = R()  
  
# Определение базового класса Stick  
class Stick:  
 def \_\_init\_\_(self, owner, length):  
 self.\_owner = owner # Инкапсуляция: приватное поле \_owner  
 if length < 0:  
 raise ValueError("Длина палки не может быть отрицательной")  
 self.\_length = length  
  
 # Переопределение метода для информативного вывода  
 def \_\_str\_\_(self):  
 return f"Owner: {self.\_owner}, Length: {self.\_length}"  
  
 # Статический метод, не требующий доступа к экземпляру  
 @staticmethod  
 def who\_are\_you():  
 print("stick")  
  
 # Метод доступа к приватному полю \_owner через декораторы  
 @property  
 def owner(self):  
 return self.\_owner  
  
 @owner.setter  
 def owner(self, owner):  
 self.\_owner = owner  
  
# Определение класса Hand, наследующего от класса Stick  
class Hand(Stick):  
 def \_\_init\_\_(self, power, \*args):  
 super().\_\_init\_\_(\*args) # Вызов конструктора родительского класса  
 self.\_power = power  
  
 # Переопределение метода для информативного вывода  
 def \_\_str\_\_(self):  
 return super().\_\_str\_\_() + f", Power: {self.\_power}"  
  
 # Полиморфный метод  
 @staticmethod  
 def who\_are\_you():  
 print("hand")  
  
 # Перегрузка оператора +  
 def \_\_add\_\_(self, other):  
 return self.\_power + other.\_power  
  
# Определение абстрактного класса Class1 с абстрактным методом test  
class Class1(metaclass=ABCMeta):  
 @staticmethod  
 @abstractmethod  
 def test(x):  
 pass  
  
# Определение класса Class2, наследующего от Class1, с виртуальным методом test  
class Class2(Class1):  
 @staticmethod  
 def test(x):  
 print(x)  
  
# Определение пользовательского исключения  
class CustomException(Exception):  
 def \_\_init\_\_(self, message="Пользовательское исключение"):  
 self.message = message  
 super().\_\_init\_\_(self.message)  
  
# Основная программа  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 # Создание и использование объектов класса Stick  
 st = Stick("Maiki", 100)  
 st.owner = "Maiki Maikovich"  
 print(st.owner)  
 print(st)  
 Stick.who\_are\_you()  
  
 print()  
  
 # Попытка создать палку с отрицательной длиной  
 stick\_with\_negative\_length = Stick("Negative Stick", -10)  
 except ValueError as ve:  
 print(f"Произошла ошибка: {ve}")  
 finally:  
 print("Конец программы")  
  
 # Использование assert  
 x = 10  
 assert x > 0, "x должен быть положительным числом"  
  
 # Генерация пользовательского исключения  
 try:  
 raise CustomException("Сработало пользовательское исключение")  
 except CustomException as ce:  
 print(ce)

Результат

