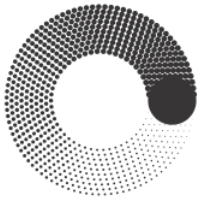
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



Московский политехнический университет

Факультет Информационных технологий

Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

Лабораторная работа № 4

Дисциплина: Backend-разработка

Тема: Исключения

Выполнил: студент группы 231-333

Фабрикант Тимур Романович
(Фамилия И.О)

Дата, подпись _____ _____
(Дата) (Подпись)

Проверил: Полубояринова А.С. ___ _____
(Фамилия И.О., степень, звание) (Оценка)

Замечание: _____

_____

Москва

2025

Цель: Ознакомиться с принципами обработки исключений

Задачи:

1. Минимум 2 разные функции, которые принимают на вход один или несколько параметров.
   Функции ДОЛЖНЫ выбрасывать исключение при определённых значениях входных параметров.
   Функции НЕ ДОЛЖНЫ содержать никаких обработчиков исключений.
2. Функция, которая принимает на вход один или несколько параметров.
   Функция ДОЛЖНА выбрасывать исключение при определённых значениях входных параметров.
   Функция ДОЛЖНА содержать ОДИН обработчик исключений общего типа (Exception). Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
   Обработчик НЕ ДОЛЖЕН содержать блок finally.
3. Функция, которая принимает на вход один или несколько параметров.
   Функция ДОЛЖНА выбрасывать исключение при определённых значениях входных параметров.
   Функция ДОЛЖНА содержать ОДИН обработчик исключений общего типа (Exception). Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
   Обработчик ДОЛЖЕН содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.
4. Минимум 3 разные функции, которые принимают на вход один или несколько параметров.
   Функции ДОЛЖНЫ выбрасывать исключения при определённых значениях входных параметров.
   Функции ДОЛЖНЫ содержать НЕСКОЛЬКО обработчиков РАЗНЫХ типов исключений (минимум 3 типа исключений). Внутри блоков обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой соответствующего типа исключения.
   Каждый обработчик МОЖЕТ содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.
5. Функция, которая принимает на вход один или несколько параметров.
   Функция ДОЛЖНА генерировать исключения при определённых условиях (в Python есть конструкция для генерации исключений).
   Функция ДОЛЖНА содержать обрабопчики всех исключений, которые генерируются внутри этой функции. Внутри блоков обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой соответствующего типа исключения.

Обработчик МОЖЕТ содержать блок finally. Логика внутри блока finally ДОЛЖНА способствовать нормальному завершению работы функции.

6. Минимум 3 разных пользовательских исключения и примеры их использования
7. Функция, которая принимает на вход один или несколько параметров. Функция ДОЛЖНА выбрасывать пользовательское исключение, созданное на шаге 6. при определённых значениях входных параметров.
   Функция ДОЛЖНА содержать МИНИМУМ ОДИН обработчик исключений. Внутри блока обработки исключения ДОЛЖНА быть какая-нибудь логика, связанная с обработкой исключения.
   Обработчик МОЖЕТ содержать блок finally.
8. Минимум 3 функции, демонстрирующие работу исключений. Алгоритм функций необходимо придумать самостоятельно
9. Функция, которая последовательно вызывает ВСЕ вышесозданные функции.
   Функция ДОЛЖНА завершаться корректно и НЕ ДОЛЖНА иметь необработанных исключений

Ссылка на код:

https://github.com/DurkaVerder/Backend-python-Polytech/tree/main/Lab4

Ход выполнения работы:

Написал классы пользовательских ошибок

Листинг 1. exceptions.py

```
class UserNotFoundError(Exception):
    """Пользователь не найден."""


class BookNotAvailableError(Exception):
    """Книга недоступна для выдачи."""


class PermissionDeniedError(Exception):
    """У пользователя нет прав на выполнение действия."""
```

Написал функции для создания и обработки исключений

Листинг 2. others.py

```
from exceptions import UserNotFoundError, BookNotAvailableError,
PermissionDeniedError


# Шаг 1
def delete_book(books: list, role: str):
    if role != "admin":
        raise PermissionDeniedError("Access denied. Only admins can delete
books.")
    if not books:
        raise ValueError("No books to delete.")
    return f"Book '{books.pop()}' deleted."


# Шаг 1
def get_book(book_id: int):
    if book_id <= 0:
        raise ValueError("Invalid book ID. Must be positive.")
    if book_id > 1000:
        raise LookupError("Book not found in database.")
    return {"id": book_id, "title": "Sample Book", "author": "Unknown"}


# Шаг 2
def get_all_books_by_author(author: str):
    try:
        if not author:
            raise ValueError("Author name cannot be empty.")
        return [
            {"id": 1, "title": "Book One", "author": author},
            {"id": 2, "title": "Book Two", "author": author},
```

```
        ]
    except Exception as e:
        print(f"An error occurred in get_all_books_by_author: {e}")
        return []


# Шаг 3
def add_book(title: str, author: str, year: int):
    try:
        if not title or not author:
            raise ValueError("Title and author cannot be empty.")
        if year < 0:
            raise ValueError("Year cannot be negative.")
        return {"id": 123, "title": title, "author": author, "year": year}
    except Exception as e:
        print(f"An error occurred in add_book: {e}")
        return None
    finally:
        print("add_book function executed.")


# Шаг 4
def check_registered_user(user_id: int):
    try:
        if user_id <= 0:
            raise ValueError("Invalid user ID. Must be positive.")
        if user_id % 5 == 0:
            raise PermissionDeniedError("User is banned.")
        if user_id > 1000:
            raise UserNotFoundError("User not found.")
        return True
    except ValueError as e:
        print(f"Validation error: {e}")
        return False
    except UserNotFoundError as e:
        print(f"User error: {e}")
        return False
    except PermissionDeniedError as e:
        print(f"Permission denied error: {e}")
        return False
    finally:
        print("check_registered_user finished.")


# Шаг 4
def borrow_book(user: str, book_id: int):
    try:
        if not user:
            raise UserNotFoundError("User must be provided.")
        if book_id <= 0:
            raise ValueError("Invalid book ID.")
```

```
        if book_id % 2 == 0:
            raise BookNotAvailableError("This book is already borrowed.")
        return f"Book {book_id} borrowed by {user}."
    except UserNotFoundError as e:
        print(f"Borrow error (user): {e}")
        return None
    except ValueError as e:
        print(f"Borrow error (validation): {e}")
        return None
    except BookNotAvailableError as e:
        print(f"Borrow error (availability): {e}")
        return None
    finally:
        print("borrow_book finished.")


# Шаг 4
def update_book_info(book_id: int, title: str = None):
    try:
        if book_id <= 0:
            raise ValueError("Book ID must be positive.")
        if book_id > 500:
            raise LookupError("Book not found.")
        if title is None:
            raise AttributeError("Title must be provided for update.")
        return f"Book {book_id} updated with new title '{title}'."
    except ValueError as e:
        print(f"Update error (validation): {e}")
    except LookupError as e:
        print(f"Update error (lookup): {e}")
    except AttributeError as e:
        print(f"Update error (attribute): {e}")
    finally:
        print("update_book_info finished.")


# Шаг 5
def reserve_book(user: str, book_id: int):
    try:
        if not user:
            raise UserNotFoundError("User is required.")
        if book_id <= 0:
            raise ValueError("Book ID must be valid.")
        if book_id % 3 == 0:
            raise BookNotAvailableError("Book cannot be reserved.")
        return f"Book {book_id} reserved for {user}."
    except UserNotFoundError as e:
        print(f"Reserve error (user): {e}")
    except ValueError as e:
        print(f"Reserve error (validation): {e}")
    except BookNotAvailableError as e:
```

```
        print(f"Reserve error (availability): {e}")
    finally:
        print("reserve_book finished.")


# Шаг 7
def validate_role(role: str):
    try:
        if role != "admin":
            raise PermissionDeniedError("Only admin can perform this action.")
        return True
    except PermissionDeniedError as e:
        print(f"Permission error: {e}")
        return False
    finally:
        print("validate_role finished.")


# Шаг 8
def search_book_by_title(title: str):
    if not title:
        raise ValueError("Title cannot be empty.")
    return f"Book with title '{title}' found."


# Шаг 8
def return_book(user: str, book_id: int):
    if not user:
        raise UserNotFoundError("User required for return.")
    if book_id <= 0:
        raise ValueError("Invalid book ID.")
    return f"Book {book_id} returned by {user}."


# Шаг 8
def recommend_books(user: str, genre: str):
    if not user:
        raise UserNotFoundError("User must be logged in.")
    if not genre:
        raise ValueError("Genre must be provided.")
    return [f"{genre} Book 1", f"{genre} Book 2"]
```

Написал функцию, которая вызывает все созданные функции из шагов 1 – 8

Листинг 3. main.py

```
from others import *


def run_all():
    print("--- Step 1 ---")
    try:
        print(delete_book(["Book1"], "admin"))
        print(get_book(10))
    except Exception as e:
        print(f"Step 1 error in run_all: {e}")

    print("\n--- Step 2 ---")
    print(get_all_books_by_author("Tamerlan"))

    print("\n--- Step 3 ---")
    print(add_book("Go", "Pablo", 1991))

    print("\n--- Step 4 ---")
    check_registered_user(10)
    print(borrow_book("Alex", 3))
    print(update_book_info(100, "New Title"))

    print("\n--- Step 5 ---")
    print(reserve_book("Alex", 7))

    print("\n--- Step 7 ---")
    validate_role("viewer")

    print("\n--- Step 8 ---")
    try:
        print(search_book_by_title("Python Advanced"))
        print(return_book("Nikita", 42))
        print(recommend_books("Semes", "Fantasy"))
    except Exception as e:
        print(f"Step 8 error in run_all: {e}")


if __name__ == "__main__":
    run_all()
```