📜 Краткая справка: Обработка ошибок в ООП

🔹 Что такое исключение (Exception)?

Исключение — это объект, который сигнализирует о возникновении ошибки или нестандартной ситуации во время выполнения программы.

Примеры встроенных исключений:

- `ValueError` — недопустимое значение
- `TypeError` — неверный тип
- `ZeroDivisionError` — деление на ноль

🔹 Как создать своё исключение?

Создайте класс, наследующий от `Exception` (или его подкласса):

```
class InvalidAgeError(Exception):
    pass

# Или с сообщением по умолчанию:
class InsufficientFundsError(Exception):
    def __init__(self, message="Недостаточно средств"):
        self.message = message
        super().__init__(self.message)
```

🔹 Как вызывать исключение?

Используйте `raise`:

```
if age < 0:
    raise InvalidAgeError("Возраст не может быть отрицательным")
```

🔹 Как обрабатывать исключение?

Используйте `try...except`:

```
try:
    account.withdraw(1000)
except InsufficientFundsError as e:
    print(f"Ошибка: {e.message}")
```

🔹 Зачем создавать свои исключения?

- Делают код **понятнее** (ошибка говорит сама за себя).
- Позволяют **точно реагировать** на конкретные ситуации.
- Улучшают **отладку и тестирование**.
- Соблюдают принцип **"явное лучше неявного"**.

🔷 Лучшие практики

- Называйте исключения с суффиксом `Error` (например, `InvalidEmailError`).
- Наследуйтесь от `Exception`, а не от `BaseException`.
- Добавляйте полезное сообщение об ошибке.
- Не перехватывайте все исключения (`except:`) — указывайте конкретный тип.

🧪 Практическое задание (60 минут): Система онлайн-банкинга с защитой от ошибок

📌 Сценарий

Вы разрабатываете систему онлайн-банкинга. В ней должны быть **чёткие правила**:

- Баланс не может быть отрицательным.
- Нельзя снять больше, чем есть на счету.
- Нельзя перевести деньги на несуществующий счёт.
- Нельзя создать счёт с отрицательным начальным балансом.

Ваша задача — реализовать класс `BankAccount` и **создать собственные классы исключений** для каждой из этих ситуаций. Затем — продемонстрировать их обработку.

🎯 Часть 1: Создание пользовательских исключений (10 минут)

Создайте **четыре класса исключений**, наследующих от `Exception`:

1. `InvalidAmountError` — вызывается, если сумма ≤ 0.
2. `InsufficientFundsError` — вызывается при попытке снять/перевести больше, чем есть на балансе.
3. `NegativeBalanceError` — вызывается, если кто-то пытается установить отрицательный баланс.
4. `AccountNotFoundError` — вызывается при переводе на несуществующий счёт.

> 💡 Каждый класс должен принимать **понятное сообщение об ошибке** (можно через `super().__init__(message)`).

🎯 Часть 2: Класс `BankAccount` (25 минут)

Атрибуты:

- `account_id` (str) — уникальный ID счёта (например, `"ACC-1001"`)
- `owner` (str) — имя владельца

- `balance` (float) — текущий баланс

Методы:

1. `__init__(self, account_id, owner, initial_balance=0.0)`
    - Если `initial_balance < 0` → вызывает `NegativeBalanceError`
    - Иначе устанавливает баланс.
2. `deposit(self, amount)`
    - Если `amount <= 0` → `InvalidAmountError`
    - Иначе увеличивает баланс.
3. `withdraw(self, amount)`
    - Если `amount <= 0` → `InvalidAmountError`
    - Если `amount > balance` → `InsufficientFundsError`
    - Иначе уменьшает баланс.
4. `transfer(self, amount, recipient_account)`
    - Если `recipient_account is None` → `AccountNotFoundError`
    - Иначе:
        - Снимает `amount` с текущего счёта (`self.withdraw`)
        - Добавляет `amount` на счёт получателя (`recipient_account.deposit`)

> 💡 Все проверки — через `raise` соответствующих исключений.

🎯 Часть 3: Демонстрация и обработка ошибок (25 минут)

Создайте функцию `simulate_banking()`:

1. Создайте два счёта:
    - `acc1 = BankAccount("ACC-1001", "Анна", 1000.0)`
    - `acc2 = BankAccount("ACC-1002", "Борис", 500.0)`
2. Выполните серию операций **в блоках `try...except`**:

Операция	Ожидаемое исключение
`acc1.deposit(-100)`	`InvalidAmountError`
`acc1.withdraw(2000)`	`InsufficientFundsError`
`acc1.transfer(300, None)`	`AccountNotFoundError`
`acc1.transfer(200, acc2)`	Успех (без ошибки)

3. Для каждой операции выведите:
    - Если успех: `"✅ Успешно: ..."`

- Если ошибка: "❌ Ошибка: {сообщение из исключения}"

4. В конце выведите финальные балансы обоих счетов.

📥 Пример ожидаемого вывода

```
❌ Ошибка: Сумма должна быть положительной
❌ Ошибка: Недостаточно средств для снятия 2000.0 руб.
❌ Ошибка: Счёт получателя не найден
✅ Успешно: Переведено 200.0 руб. на счёт ACC-1002

Финальные балансы:
ACC-1001 (Анна): 800.0 руб.
ACC-1002 (Борис): 700.0 руб.
```

📝 Требования к сдаче

- Один файл `.py`.
- Четыре класса исключений.
- Класс `BankAccount` с полной логикой.
- Функция `simulate_banking()` с демонстрацией.
- Все исключения **обрабатываются**, программа не падает.
- Сообщения об ошибках — понятные и на русском языке.

💡 Подсказки

- В `transfer` сначала проверяйте получателя, **потом** вызывайте `withdraw` и `deposit`.
- В исключениях можно передавать детали:

```
raise InsufficientFundsError(f"Недостаточно средств: баланс {self.balance}, запрошено {amount}")
```

- Используйте `except InvalidAmountError as e: print(f"Ошибка: {e}")` — сообщение будет из `super().__init__`.
