**итоговая практика на 1 час**, объединяющая **все ключевые темы ООП**, изученные за курс:

- Классы и объекты
- Атрибуты и методы экземпляра
- Наследование и полиморфизм
- Инкапсуляция (`_`, `__`, `@property`)
- Магические методы (`__str__`, `__eq__`, `__lt__`)
- Обработка ошибок → пользовательские исключения
- Паттерны проектирования → `@classmethod` (альтернативный конструктор), `@staticmethod` (валидация)

---

# 🧪 Итоговая практика (60 минут):

## Система учёта сотрудников IT-компании

> ✅ **Цель**: Создать комплексную модель HR-системы, в которой:
>
> - Есть разные типы сотрудников (наследование + полиморфизм)
> - Данные защищены и проходят валидацию (инкапсуляция + `@property`)
> - Объекты можно сравнивать и сортировать (магические методы)
> - Ошибки обрабатываются через собственные исключения
> - Есть альтернативный способ создания сотрудника (из строки)

---

## 📚 Краткий обзор того, что будет использовано

| Тема | Где применяется в задании |
|---|---|
| **Наследование** | `Employee` → `Developer`, `Designer`, `Manager` |
| **Полиморфизм** | Метод `calculate_bonus()` работает по-разному для каждого типа |
| `@property` | Защита атрибутов `name`, `salary` с валидацией |
| **Магические методы** | `__str__` — для вывода, `__eq__` и `__lt__` — для сравнения |
| **Исключения** | `InvalidSalaryError`, `InvalidNameError` |
| `@classmethod` | `from_string()` — альтернативный конструктор |
| `@staticmethod` | `validate_name()`, `validate_salary()` — валидация |

---

## 🎯 Задание: Реализуйте классы и продемонстрируйте работу

### 🔷 Шаг 1: Создайте пользовательские исключения (5 минут)

Создайте два класса исключений:

```
class InvalidNameError(Exception):
    pass

class InvalidSalaryError(Exception):
    pass
```

---

## ◈ Шаг 2: Базовый класс `Employee` (20 минут)

**Атрибуты** (все защищённые):

- `_name` — имя сотрудника
- `_salary` — оклад (до бонусов)
- `_position` — должность (устанавливается в дочерних классах)

**Методы**:

1. **Конструктор** `__init__(name, salary)`:
   - Валидация через статические методы (см. ниже).
   - Устанавливает `_name`, `_salary`.
2. **Свойства (`@property`)**:
   - `name` — только для чтения.
   - `salary` — геттер и сеттер с валидацией (через `validate_salary`).
3. **Абстрактный по смыслу метод**:
   - `calculate_bonus()` — должен быть переопределён в потомках. Возвращает `float`.
4. **Магические методы**:
   - `__str__()` → `"Имя (Должность): оклад XXX, бонус YYY"`
   - `__eq__(self, other)` → равны, если **общий доход** (оклад + бонус) равен.
   - `__lt__(self, other)` → меньше, если **общий доход** меньше.
5. **Статические методы**:
   - `validate_name(name)` → `True`, если строка непустая, без цифр.
   - `validate_salary(salary)` → `True`, если число и >= 30 000.
6. **Метод класса**:
   - `from_string(cls, emp_str)` → создаёт объект из строки формата `"Имя - Оклад"`
     Пример: `"Анна - 80000"`

---

## ◈ Шаг 3: Дочерние классы (15 минут)

**а)** `Developer(Employee)`

- `_position = "Разработчик"`
- `calculate_bonus()` → `0.1 * _salary` (10% от оклада)

**b)** `Designer(Employee)`

- `_position = "Дизайнер"`
- `calculate_bonus()` → `0.15 * _salary` (15%)

**c)** `Manager(Employee)`

- `_position = "Менеджер"`
- `calculate_bonus()` → `0` (без бонуса)

> 💡 В `__init__` вызывайте `super().__init__(name, salary)` и устанавливайте `_position`.

---

## 🔷 Шаг 4: Демонстрация (20 минут)

Напишите функцию `run_hr_system()`:

1. Создайте 4 сотрудника:
   - Обычно: `dev = Developer("Анна", 80000)`
   - Через строку: `mgr = Manager.from_string("Борис - 120000")`
   - Ещё два — разного типа
2. Поместите их в список.
3. Для каждого выведите через `print()` (используется `__str__`).
4. Отсортируйте список по общему доходу (используется `__lt__`).
5. Найдите самого высокооплачиваемого.
6. Проверьте, есть ли сотрудники с одинаковым доходом (используется `__eq__`).
7. Протестируйте ошибки:
   - `Developer("123", 50000)` → `InvalidNameError`
   - `Designer("Катя", 20000)` → `InvalidSalaryError`

   Оберните в `try...except` и выведите сообщение.

📌 **Пример вывода**:

```
Анна (Разработчик): оклад 80000, бонус 8000.0
Борис (Менеджер): оклад 120000, бонус 0
Катя (Дизайнер): оклад 70000, бонус 10500.0

=== После сортировки ===
Катя (Дизайнер): оклад 70000, бонус 10500.0
Анна (Разработчик): оклад 80000, бонус 8000.0
```

```
Борис (Менеджер): оклад 120000, бонус 0

Самый высокооплачиваемый: Борис
Есть сотрудники с одинаковым доходом: False

Ошибки:
❌ Некорректное имя: 123
❌ Некорректный оклад: 20000
```

---

## Требования к сдаче

- Один файл `.py`
- Все классы и функции
- Демонстрационный код в `if __name__ == "__main__":`
- Никакого `input()`
- Все темы ООП применяются осознанно

---

## Советы по реализации

- В `__str__` используйте `self.calculate_bonus()`.
- В магических методах тоже вызывайте `calculate_bonus()`, чтобы не хранить кэш.
- В `from_string` проверяйте формат строки и валидируйте данные.
- Для проверки одинаковых доходов:

```
has_dup = any(e1 == e2 for i, e1 in enumerate(employees) for e2 in
employees[i+1:])
```