Теоретическая сводка

Что нужно знать:

- Интерфейсы могут расширять **другие интерфейсы** (множественное наследование интерфейсов возможно, но не классов).
- В интерфейсах могут быть `default`, `static`, `private` методы, а также константы.
- Конфликт `default`-методов нужно разрешать переопределением.
- Можно создавать **generic-интерфейсы**, где типы параметризуются, например `interface Repository<T>`.
- Интерфейсы могут быть использованы для реализации **паттернов проектирования** (Strategy, Observer, Factory).

5 Заданий по Интерфейсам (Java)

📌 Задание 1 — Система оплаты заказов (несколько интерфейсов)

Теория:
Класс может реализовать несколько интерфейсов, что позволяет комбинировать функциональность.
Если методы совпадают по сигнатуре, нужно реализовать один общий метод.

Задание:
Создайте систему для обработки заказов.

- `interface Payable` — метод `processPayment(double amount)`
- `interface Refundable` — метод `processPayment(double amount)` (именно с тем же именем, чтобы студенты столкнулись с дублированием).
- `class Order` должен реализовать оба интерфейса, но внутри `processPayment()` выводить разные сообщения в зависимости от того, платёж это или возврат.

Подсказка: студенты должны хранить тип операции в классе (например, флаг `isRefund`).

📌 Задание 2 — Игра с персонажами (вложенный интерфейс)

Теория:
Интерфейсы могут содержать **вложенные интерфейсы**. Это удобно для описания сложных систем.

Задание:
Сделайте:

- `interface Character` с методом `act()`.
- Внутри него вложенный интерфейс `Inventory` с методом `listItems()`.
- Создайте класс `Warrior`, который реализует **оба интерфейса** (внешний и вложенный).
- В методе `act()` пусть он атакует, а в `listItems()` выводит список предметов (например, "Меч, Щит").

📌 Задание 3 — Абстрактный класс + интерфейс: расчёт зарплаты

Теория:
Абстрактные классы могут частично реализовывать интерфейсы, а конкретные классы доопределяют недостающие методы.

Задание:

- `interface Employee` с методами `double calculateSalary()` и `String getRole()`.
- Абстрактный класс `BaseEmployee` реализует только `getRole()` (например, возвращает "Employee").
- Создайте `Developer` (поле `hourlyRate`, `hoursWorked`), который расширяет `BaseEmployee` и реализует `calculateSalary()`.
- Пусть main создаёт разработчика и печатает его роль и зарплату.

📌 Задание 4 — Сложный полиморфизм: экспорт данных

Теория:
Интерфейсы часто используют, чтобы подставлять разные реализации в один и тот же метод.

Задание:

- Создайте интерфейс `Exporter` с методом `export(String data)`.
- Создайте два класса:
  - `JsonExporter` (выводит `{ "data": "..." }`)
  - `XmlExporter` (выводит `<data>...</data>`)
- Создайте класс `ReportGenerator` с методом `generate(Exporter exporter)`, который формирует отчёт и передаёт его на экспорт.
- В `main()` вызовите генератор с разными экспортёрами.

📌 Задание 5 — Интерфейсы + константы + статические методы

Теория:
В интерфейсах можно объявлять константы и **static методы**. Это полезно для утилит.

Задание:

- `interface MathUtils` с константой `PI` и статическим методом `double circleArea(double r)`.
- Создайте класс `Circle`, который хранит радиус и в методе `printArea()` вызывает `MathUtils.circleArea(r)`.
- В `main()` создайте несколько кругов и выведите их площади.