**Лабораторная работа №3. TypeScript**

**1. Классы и ООП**

**Задача 1: Система управления пользователями**

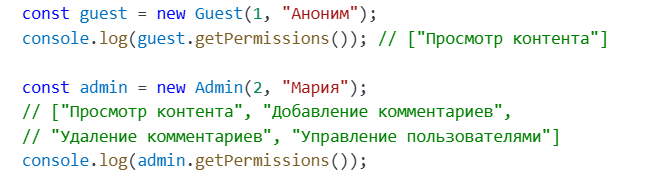
Реализуйте систему пользователей, в которой есть три типа:

1. Guest — может только просматривать контент.
2. User — может оставлять комментарии.
3. Admin — может удалять комментарии и управлять пользователями.

**Требования:**

* Создайте абстрактный класс BaseUser с полями id, name и методом getRole(): string.
* Создайте классы Guest, User, Admin, наследуемые от BaseUser.
* Реализуйте метод getPermissions(), который возвращает список доступных действий для каждой роли.

**Пример использования:**

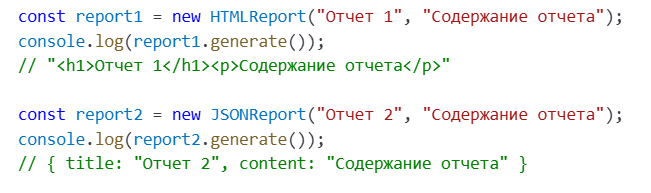


**Задача 2: Полиморфизм и интерфейсы**

Реализуйте систему отчетов, где:

* У каждого отчета (Report) есть title, content и метод generate().
* Есть два типа отчетов:
  + HTMLReport (возвращает HTML-код)
  + JSONReport (возвращает JSON-объект)
* Используйте интерфейс IReport.

**Пример использования:**



**2. Дженерики (Generics)**

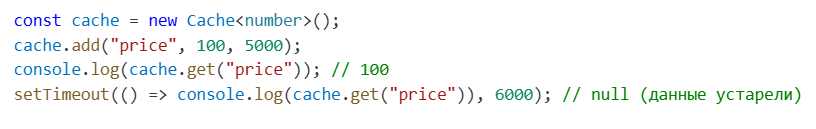
**Задача 3: Обобщенный кеш данных**

Создайте класс Cache<T>, который позволяет хранить данные в кеше на определенное время.

**Требования:**

* add(key: string, value: T, ttl: number) — добавляет элемент с временем жизни ttl (в мс).
* get(key: string): T | null — возвращает значение или null, если время истекло.
* clearExpired() — очищает просроченные элементы.

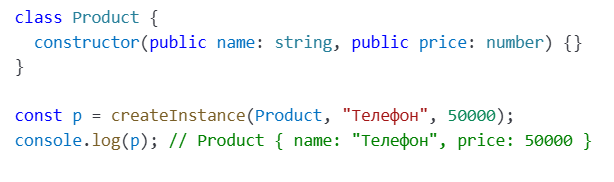
**Пример использования:**



**Задача 4: Дженерик-фабрика объектов**

Создайте функцию createInstance<T>(cls: new (...args: any[]) => T, ...args: any[]): T, которая принимает конструктор класса и аргументы, а затем создает новый объект.

**Пример использования:**



**3. Кортежи и Enum**

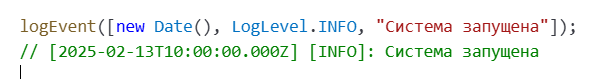
**Задача 5: Логирование событий с кортежами**

Создайте систему логирования, где каждое событие представляется кортежем [timestamp, level, message].

**Требования:**

* timestamp — дата в формате Date.
* level — enum LogLevel с уровнями (INFO, WARNING, ERROR).
* message — строка.
* Реализуйте функцию logEvent(event: LogEntry), которая принимает кортеж и выводит его в консоль.

**Пример использования:**



**Задача 6: Тип безопасных API-ответов**

Реализуйте enum HttpStatus с кодами 200, 400, 401, 500 и тип ApiResponse<T>, содержащий кортеж [status: HttpStatus, data: T | null, error?: string].

**Требования:**

* success<T>(data: T): ApiResponse<T> — создает успешный ответ (200).
* error(message: string, status: HttpStatus): ApiResponse<null> — создает ошибочный ответ.

**Пример использования:**

