🎬 СЦЕНАРИЙ ВИДЕО: "КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ 'МЕНЕДЖЕР РАСХОДОВ' НА JAVA С НУЛЯ"

ЧАСТЬ 1: ВВЕДЕНИЕ (3 минуты)

Тайминг	**Текст**	**Акценты и Визуализация**
0:00-3:00	"Привет, друзья! Сегодня мы создадим с нуля полезное консольное приложение **'Менеджер Расходов' на Java**."	Заставка, показывающая три ключевых класса.
	"Это идеальный проект для начинающих, который позволит вам перейти от основ процедурного кода к **объектно-ориентированному программированию (ООП)**, научит работать с **коллекциями** объектов, пользовательским вводом и, самое главное, с **файловым вводом/выводом (I/O)** для **сохранения финансовой истории**!"	Выделение ключевых слов (ООП, I/O).
	"Менеджер Расходов — это отличный способ прокачать навыки Java, научиться хранить **структурированные данные** (сумма, категория, дата) в памяти и на диске, правильно **разделять логику** через классы и создавать **персистентные** приложения."	Краткая демонстрация работающей программы (пользователь добавляет расходы, считает итог, закрывает, открывает, история на месте).
	Наше приложение будет уметь:	
	* Принимать выбор пользователя через **консольное меню**.	
	* **Добавлять** новые расходы (Сумма, Категория, Дата).	
	* **Просматривать** весь список расходов с датами.	
	* **Подсчитывать общую сумму** расходов (ИТОГ).	
	* **Сохранять и загружать** данные расходов из текстового файла.	
	* **Обрабатывать некорректный ввод** пользователя (например, неверный формат суммы или даты).	

Тайминг	Текст	Акценты и Визуализация
	"Всё это мы напишем в трех классах: **Expense**, **ExpenseManager** и **ExpenseApp (Main)**, чтобы вы увидели, как организовать структуру настоящего приложения!"	

ЧАСТЬ 2: ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ (3 минуты)

Тайминг	Текст	Акценты и Визуализация
3:00-6:00	"Мы используем только стандартные возможности Java и ее библиотеки I/O, включая современные возможности для работы с датами и файлами."	Краткий показ используемых importов.
	Что применим:	
	* **Классы ООП:** Expense, ExpenseManager, ExpenseApp для четкой структуры.	
	* **Scanner** для интерактивного взаимодействия с пользователем и получения ввода.	
	* **ArrayList<Expense>** (интерфейс List) для хранения записей расходов в памяти.	
	* **LocalDate** для надежной работы с датами.	
	* **FileWriter, PrintWriter** для сохранения данных в файл (expenses.txt).	
	* **FileReader, BufferedReader** для загрузки данных из файла.	
	* Циклы **while** для основного меню и **оператор switch** для чистой реализации меню.	
	* **Stream API (Java 8+)** для эффективного и современного **подсчета общей суммы** и **сортировки** расходов.	
	* Блок **try-with-resources** для безопасной работы с файлами и предотвращения утечек.	

Тайминг	Текст	Акценты и Визуализация
	"Мы создадим компактное, но полнофункциональное приложение, которое демонстрирует все ключевые концепции начального ООП и персистентности на Java."	

ЧАСТЬ 3: ЧЕМУ ВЫ НАУЧИТЕСЬ (3 минуты)

Тайминг	Текст	Акценты и Визуализация
6:00-9:00	"После просмотра видео вы получите не только работающий Менеджер Расходов, но и глубокое понимание фундаментальных концепций Java."	
	Вы поймёте:	
	* Как применять **принципы ООП (инкапсуляция)** на практике через класс Expense.	
	* Как управлять состоянием приложения и централизовать логику в классе (ExpenseManager).	
	* Как работать с коллекциями (List/ArrayList) для хранения пользовательских объектов.	
	* Как использовать **файловый ввод/вывод** для сохранения данных между запусками программы (**персистентность**).	
	* Как использовать **Stream API** для эффективного **суммирования** и **фильтрации** данных.	
	* Как **обрабатывать ошибки ввода** (например, NumberFormatException, DateTimeParseException) и исключения при работе с файлами.	
	* Как строить главный цикл приложения и реализовывать меню через switch.	
	"Это идеальный стартовый проект для практики структурного проектирования и основ персистентности данных на Java!"	

ЧАСТЬ 4: ФУНКЦИОНАЛ ПРИЛОЖЕНИЯ (3 минуты)

Тайминг	Текст	Акценты и Визуализация
9:00-12:00	"Мы создадим полноценную систему финансового учета с удобным интерфейсом и надежным хранением!"	Показ консоли с работающим приложением.
	Что будет у нашей программы:	
	* **Интуитивное меню выбора** (1. Добавить, 2. Просмотреть, 3. Итог, 4. Сохранить, 0. Выход).	
	* **ООП-Архитектура**: четкое разделение ответственности между классами (Модель, Менеджер, Интерфейс).	
	* **Персистентность данных**: записи сохраняются в файл expenses.txt и загружаются при запуске.	
	* **Финансовый анализ**: быстрый расчет **общей суммы расходов**.	
	* **Система Валидации**: проверка на корректность формата суммы и даты (YYYY-MM-DD) при добавлении.	
	* **Безопасный I/O**: использование try-with-resources для гарантии закрытия файловых потоков.	
	"Мы создадим систему, которая не только работает, но и использует лучшие практики современного Java-кода!"	

ЧАСТЬ 5: НАПИСАНИЕ КОДА (15 минут)

Тайминг	Текст	Акценты и Визуализация
12:00-27:00	"Всё максимально понятно и структурировано. Мы не просто пишем код, мы разбираем его логику и алгоритмы."	
	Блок 1: Модель Данных (Expense.java)	

Тайминг	Текст	Акценты и Визуализация
	* Создаём класс Expense и **приватные поля** amount, category и date (LocalDate).	Построчный ввод кода Expense.java.
	* Пишем **Конструктор** для инициализации объекта с парсингом строки даты.	
	* Реализуем **Геттеры** для безопасного доступа к полям (**Инкапсуляция**).	
	* Переопределяем метод **toString()** для красивого табличного консольного вывода.	
	* Создаём метод **toFileString()** для подготовки строки к сохранению (разделение ,).	
	Блок 2: Менеджер и Бизнес-логика (ExpenseManager.java)	Построчный ввод кода ExpenseManager.java.
	* Создаём класс ExpenseManager с приватной коллекцией **List<Expense>**.	
	* Пишем методы **addExpense** и **viewExpenses**.	
	* Реализуем метод **calculateTotal** с использованием **Stream API** (mapToDouble и sum).	Визуализация работы Stream.
	* Вызываем **loadExpenses()** в конструкторе, чтобы обеспечить персистентность.	
	Блок 3: Файловый Ввод/Вывод (I/O)	Построчный ввод методов I/O.
	* Реализуем **saveExpenses()**: Используем **PrintWriter** внутри **try-with-resources**.	
	* Реализуем **loadExpenses()**: Используем **BufferedReader** внутри **try-with-resources**.	
	* Добавляем логику **парсинга строки** (split(",")) и **проверки типов данных** при загрузке.	
	Блок 4: Главный класс и Интерфейс (ExpenseApp.java)	Построчный ввод кода ExpenseApp.java.

Тайминг	Текст	Акценты и Визуализация
	* Создаём класс ExpenseApp и метод main.	
	* Инициализируем **Scanner** и **ExpenseManager**.	
	* Реализуем метод **displayMenu()**.	
	* Организуем главный цикл **while(running)**.	
	* Используем **switch** для обработки выбора пользователя.	
	* Добавляем **валидацию ввода** (обработка InputMismatchException) и обработку ошибок формата (NumberFormatException, DateTimeParseException).	
	"Каждый блок мы будем писать и сразу же тестировать, чтобы вы видели результат каждого этапа!"	

ЧАСТЬ 6: ЗАВЕРШЕНИЕ (3 минуты)

Тайминг	Текст	Акценты и Визуализация
27:00-30:00	"Подводим итоги: Мы научились создавать интерактивные консольные приложения на Java с использованием **ООП**, **персистентности данных** и современных инструментов, таких как **Stream API**."	Обзор финальной версии кода.
	* На практике разобрали работу с **Scanner**, **List** и **File I/O** (PrintWriter, BufferedReader).	
	* Реализовали полноценный, функциональный менеджер расходов с системой сохранения/загрузки.	
	* Создали надежную архитектуру с четким **разделением ответственности между классами**.	
	"Этот проект — отличная основа для дальнейшего развития. Вы можете добавить функционал поиска по категории,	

Тайминг	Текст	Акценты и Визуализация
	построения отчетов за месяц или даже перевести это в **графический интерфейс**!"	
	"Если тебе понравился проект, поставь лайк и подпишись, впереди будет ещё больше практических проектов на Java!"	
	"С вами был (твоё имя), до встречи в следующем видео, где мы создадим новый полезный инструмент на Java!"	