Слайд №2:

Сегодня мы разберём паттерн проектирования Prototype, который позволяет создавать копии объектов без привязки к их конкретным классам. Этот подход особенно полезен, когда прямое создание объекта требует больших ресурсов или когда система должна оставаться гибкой к изменениям.

Представьте, что вы разрабатываете игру. Каждый раз, когда появляется новый враг, система загружает его параметры из базы данных, что занимает время. Или другой пример: у вас есть сложный UI-компонент с множеством настроек, и вам нужно создать его копию с небольшими изменениями.

Проблемы:

1. Прямое создание объекта может быть медленным.
2. Жёсткая привязка к классам усложняет поддержку кода.

Prototype решает эти проблемы, позволяя объектам клонировать самих себя.

Слайд №3:

Вместо того чтобы создавать объект заново, мы поручаем эту работу самому объекту. Он реализует метод clone(), который возвращает точную копию. Prototype относится к порождающим паттернам, применяемый к объектам.

Как это работает?

1. Клиентский код вызывает prototype.clone().
2. Объект создаёт свою копию, используя либо поверхностное, либо глубокое копирование.

Это похоже на клонирование клетки: новая клетка наследует все свойства исходной, но существует независимо."

Слайд №4:

Вот как выглядит структура паттерна:

1. Интерфейс Prototype объявляет метод clone().
2. Конкретный класс ConcretePrototype реализует этот метод, определяя, как именно будет создаваться копия.

Вот минимальный пример на Python:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Здесь deepcopy гарантирует, что все вложенные объекты тоже будут скопированы, а не просто переданы по ссылке.

Слайд №5:

"Рассмотрим практический пример. Допустим, у нас есть класс GameUnit, описывающий персонажа игры. Его клонирование будет выглядеть так.

Теперь мы можем создавать копии орка без повторной инициализации, что особенно полезно, если конструктор класса выполняет 'тяжёлые' операции."\*

Слайд №6:

\*"Важно понимать разницу между типами копирования:

1. **Поверхностное (shallow):** Копируются только 'верхние' поля. Если объект содержит ссылки, они остаются общими.
2. **Глубокое (deep):** Рекурсивно копируется весь объект, включая все вложенные объекты.

Prototype обычно использует глубокое копирование, чтобы избежать неожиданных багов."\*

Слайд №7:

\*"Паттерн Prototype — это мощный инструмент, но у него есть свои ограничения.

**Преимущества:**  
✅ Уменьшает нагрузку на систему (не нужно повторять 'дорогие' операции).  
✅ Упрощает создание объектов с сложной структурой.  
✅ Позволяет гибко настраивать копии.

**Недостатки:**  
❌ Глубокое копирование может быть медленным для больших объектов.  
❌ Требует аккуратности с изменяемыми (mutable) полями, например, списками."\*

Слайд №8:

"Где используют Prototype?

1. **Игры:** Клонирование NPC, предметов, заранее загруженных уровней.
2. **Кэширование:** Сохранение состояния объекта для быстрого восстановления.
3. **UI-библиотеки:** Шаблонные элементы (кнопки, меню) с небольшими изменениями.

Например, в Unity Prototype применяется для создания префабов — шаблонов игровых объектов.

Слайд №9:

"Итак, Prototype — это:

1. **Альтернатива** фабрикам и конструкторам.
2. **Оптимизация** для ресурсоёмких объектов.
3. **Гибкость** при работе с изменяемыми состояниями.

Запомните: 'Зачем создавать, если можно клонировать?'