ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ И ИХ ПРИМЕНЕНИЕ ПРИ ПРОЕКТИРОВАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ

Торубаров Е.А., студент,

Какурина А.В., преподаватель,

ФГБОУ ВО «Юго-Западный государственный университет», г. Курск, Россия

Аннотация. В статье рассматриваются основные принципы объектно-ориентированного программирования (ООП) и их значение при проектировании информационных систем (ИС). Подробно анализируются такие ключевые концепции, как инкапсуляция, наследование, полиморфизм и абстракция, а также их влияние на структуру, гибкость, расширяемость и сопровождаемость программных решений. На примерах из практики проектирования ИС демонстрируется, как применение принципов ООП способствует снижению сложности разработки, повышению качества кода и упрощению интеграции новых функций. Особое внимание уделено вопросам моделирования предметной области, проектированию классов и объектов, а также архитектурным паттернам, основанным на ООП.

Ключевые слова: объектно-ориентированное программирование (ООП); принципы ООП; наследование; полиморфизм; абстракция; проектирование информационных систем; архитектура ПО

Автор: Торубаров Е.А., Какурина А.В.
05.05.2026 09:35 - Обновлено 05.05.2026 09:47

В современном мире информационных технологий проектирование и разработка сложных информационных систем требует не только глубоких технических знаний, но и строгого соблюдения архитектурных подходов, обеспечивающих качество, надёжность и масштабируемость конечного продукта. Одним из фундаментальных столпов современной разработки программного обеспечения является объектно-ориентированное программирование. [1]

ООП — это не просто синтаксическая надстройка или набор правил написания кода. Это парадигма мышления, которая позволяет моделировать реальный мир в виде программных сущностей — объектов. Такой подход кардинально изменил индустрию, позволив создавать системы, которые легче понимать, изменять и расширять. В данной статье мы подробно разберем четыре основных принципов ООП — инкапсуляцию, наследование, полиморфизм и абстракцию, — а также рассмотрим, как их грамотное применение на этапе проектирования закладывает фундамент для успешной реализации и долгосрочной эксплуатации информационных систем.

В основе ООП лежат четыре ключевых принципа. Их понимание и правильное применение являются залогом создания качественного объектно-ориентированного дизайна.

1. Инкапсуляция

Суть: Инкапсуляция — это механизм объединения данных (атрибутов) и методов (функций), работающих с этими данными, в единую сущность — класс. Кроме того, она предполагает сокрытие внутренней реализации объекта от внешнего мира. Доступ к данным осуществляется строго через публичные методы (интерфейс) объекта.

Цель: Защита целостности данных. Инкапсуляция предотвращает несанкционированный или случайный доступ к внутренним полям объекта, гарантируя, что объект всегда находится в корректном состоянии.

Пример: Класс Банковский Счет инкапсулирует поле баланс. Прямое изменение этого поля извне запрещено. Вместо этого класс предоставляет методы пополнить(сумма) и снять(сумма), которые содержат логику проверки (например, запрет на снятие суммы, превышающей баланс).

2. Наследование

Суть: Наследование позволяет создавать новые классы (потомки, дочерние классы) на основе уже существующих (родительских, базовых классов). Потомок наследует все поля и методы родителя, но может добавлять свои собственные или переопределять унаследованные.

Цель: Повторное использование кода и создание иерархий. Наследование реализует принцип «является» (is-a). Оно позволяет строить логические структуры от общего к частному.

Пример: Есть базовый класс Сотрудник с полями ФИО и должность. От него можно унаследовать классы Программист и Менеджер. Оба они являются сотрудниками, но у программиста могут быть свои специфические поля (например, основной_язык), а у менеджера — свои (например, количество_подчиненных).

3. Полиморфизм

Суть: Полиморфизм («многообразие форм») позволяет использовать объекты разных классов через единый интерфейс. Это означает, что один и тот же метод может по-разному реализовываться в разных классах.

Цель: Гибкость и расширяемость системы. Полиморфизм позволяет писать код, который работает с объектами на уровне их базового типа или интерфейса, не зная их конкретной реализации на момент написания кода.

Пример: У нас есть базовый класс Фигура с методом нарисовать(). Классы Круг, Квадрат и Треугольник наследуются от Фигуры и переопределяют метод нарисовать(). В коде программы можно создать список объектов типа Фигура, в котором будут храниться и круги, и квадраты. При вызове метода нарисовать() для каждого объекта из списка будет выполнена соответствующая реализация для круга или квадрата.

4. Абстракция

Суть: Абстракция — это процесс выделения наиболее значимых характеристик объекта и игнорирования несущественных деталей. В программировании она реализуется через абстрактные классы и интерфейсы, которые определяют «что» объект делает, но не «как» он это делает.

Цель: Упрощение сложной реальности. Абстракция позволяет разработчику работать

на высоком уровне, не погружаясь в детали реализации каждого компонента системы.

Пример: В системе управления автомобилем есть интерфейс Двигатель с методом запустить(). Для разработчика системы управления не важно, какой именно двигатель установлен — бензиновый или электрический. Ему достаточно знать, что у любого двигателя есть метод запуска. Конкретная реализация этого метода будет скрыта в классах БензиновыйДвигатель и ЭлектрическийДвигатель.

Применение принципов ООП при проектировании ИС

Применение этих принципов на этапе проектирования архитектуры информационной системы дает ряд неоспоримых преимуществ.

Моделирование предметной области

Первым шагом в проектировании ИС является анализ предметной области. ООП предоставляет идеальные инструменты для этого. Бизнес-сущности (клиент, заказ, товар) естественным образом моделируются в виде классов. Их свойства становятся атрибутами, а бизнес-операции — методами. Это создает «единый язык» между аналитиками, заказчиками и разработчиками. [2]

Повышение сопровождаемости кода

Системы редко остаются неизменными. Бизнес-требования эволюционируют, появляются новые функции. Код, написанный с соблюдением принципов ООП (особенно инкапсуляции), гораздо легче модифицировать. Изменение внутренней логики одного класса не требует переписывания всего приложения, если его публичный интерфейс остался прежним.

Обеспечение гибкости и расширяемости

Наследование и полиморфизм позволяют добавлять новую функциональность без изменения существующего, протестированного кода. Например, если в ИС для интернет-магазина нужно добавить новый способ оплаты (например, криптовалюту), достаточно создать новый класс-наследник от базового класса СпособОплаты и реализовать в нем специфическую логику. Основной код магазина при этом менять не нужно — он просто будет работать с новым объектом через общий интерфейс.

Снижение сложности

Абстракция позволяет разбить сложную систему на множество простых, независимых компонентов. Разработчик может сосредоточиться на реализации одного модуля (класса), зная только его интерфейс взаимодействия с другими частями системы. Это значительно снижает когнитивную нагрузку и риск внесения ошибок. [3]

Таблица 1 - Сравнительная таблица: Процедурный vs Объектно-ориентированный подход

Критерий
Процедурный подход
Объектно-ориентированный подход
Основной элемент
Функция/процедура
Объект/Класс
Организация данных
Данные отделены от объектов
Данные и функции объединены в объекте
Доступ к данным
Глобальные переменные или передача в функции

Инкапсуляция,	доступ через методы
Повторное	использование
Через	копирование кода или библиотеки функций
Через	наследование и композицию
Расширяемость	
Сложная,	требует модификации существующего кода
Простая,	через добавление новых классов-наследников
Сложность	системы
Растет	нелинейно с добавлением функций
Управляется	за счет абстракции и модульности

Пример из жизни

Рецепт (алгоритм действий)

Кухонный комбайн (объект с функциями)

Архитектурные паттерны на основе ООП

Принципы ООП лежат в основе многих популярных архитектурных паттернов, которые являются стандартом де-факто при проектировании ИС.

· MVC (Model-View-Controller):

· Model (Модель): Классы, представляющие данные бизнес-логики и правила их изменения.

· View (Представление): Классы, отвечающие за отображение данных пользователю (UI).

· Controller (Контроллер): Классы-посредники, обрабатывающие ввод пользователя, взаимодействующие с Моделью и обновляющие Представление.

· Repository (Репозиторий): Паттерн для организации слоя доступа к данным. Класс-репозиторий инкапсулирует всю логику работы с источником данных (например, базой данных), предоставляя остальной части приложения простой интерфейс для получения и сохранения объектов-сущностей.

· Factory (Фабрика): Класс-фабрика отвечает за создание объектов. Это полезно, когда процесс создания объекта сложен или когда тип создаваемого объекта определяется во время выполнения программы (полиморфное создание). [4]

Заключение

Объектно-ориентированное программирование — это мощная парадигма, которая при правильном применении становится незаменимым инструментом в руках архитектора и разработчика информационных систем. Инкапсуляция защищает данные и упрощает поддержку, наследование способствует повторному использованию кода, полиморфизм обеспечивает гибкость, а абстракция позволяет управлять сложностью.

Грамотное проектирование ИС на основе принципов ООП позволяет создавать программные продукты, которые не просто выполняют текущие задачи бизнеса, но и готовы к будущим изменениям. Такие системы легче тестировать, сопровождать и масштабировать, что в конечном итоге снижает общую стоимость владения программным обеспечением и повышает его конкурентоспособность на рынке.

Литература

1. Лафоре, Р. Объектно-ориентированное программирование в C++ [Текст] / Р. Лафоре. - 4-е изд. - СПб. [и др.] : Питер, 2026. – 928 с.
2. Зыков, С.В. Введение в теорию программирования [Электронный ресурс] / С.В. Зыков. – М. : Национальный откр ытый университет «ИНТУИТ», 2016. – 189 с. – Режим доступа / https://biblioclub.ru/
3. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами и приложениями на C++ [Текст] / Пер. с англ. - 2-е изд. - СПб. : Бином ; СПб. : Невский диалект, 2001. - 560 с.
4. Сорокин, А.А. Объектно-ориентированное программирование [Электронный ресурс] : учебное пособие (курс лекций) / А.А. Сорокин. – Ставрополь : Изд-во СКФУ, 2014. – 174 с. Режим доступа / https://biblioclub.ru/