

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  информационных технологий | **Кафедра**  информационных систем |

**Реферат № 6**

по дисциплине «**Основы программной инженерии**»

на тему: «Конструирование программного обеспечения»

|  |  |  |
| --- | --- | --- |
| **Студент** группа ИДБ–22–06 |  | **Мустафаева П.М.** |
|  | подпись |  |
| **Руководитель**  к.э.н., доцент |  | **Ахмедов Э.Р.** |
|  | подпись |  |

**1.** **Что включает в себя процесс конструирования ПО? Перечислите основные категории ПО и объясните особенности процесса конструирования для каждой из них.**

Конструирование ПО – это ключевой этап разработки, на котором проектные решения превращаются в реальный работающий код. Этот процесс включает два основных направления: низкоуровневое проектирование – детализация архитектуры перед непосредственным написанием кода; кодирование – непосредственная реализация программы на выбранном языке программирования.

Рассмотрим каждый из этих аспектов подробнее.

**Низкоуровневое проектирование.**

На этом этапе разработчики углубляются в детали системы, определяя, как именно будут работать её компоненты. В зависимости от типа ПО проектирование может включать:

* Объектно-ориентированное проектирование (ООП)
* Определение классов, их свойств и методов.
* Проработка взаимодействия между объектами (наследование, композиция, агрегация).
* Применение паттернов проектирования (Singleton, Factory, Observer и др.) для решения типовых задач.
* Проектирование базы данных
* Разработка схемы данных (таблицы, связи, индексы).
* Оптимизация запросов (SQL или NoSQL в зависимости от СУБД).
* Учет требований к масштабируемости и безопасности.
* Организация веб-приложений
* Разделение на клиентскую (frontend) и серверную (backend) части.
* Проектирование API (REST, GraphQL, gRPC).
* Выбор архитектурного стиля (MVC, микросервисы, serverless).
* Системное и встраиваемое ПО
* Работа с аппаратными ограничениями (память, процессор, энергопотребление).
* Использование низкоуровневых языков (C, Rust, ассемблер).
* Учет реального времени (RTOS, обработка прерываний).

**Кодирование.**

После завершения проектирования начинается этап написания кода. Здесь важно не просто перевести алгоритмы в программные инструкции, но и обеспечить:

* Читаемость и поддерживаемость кода
* Следование code style (PEP 8 для Python, Google Style Guide для C++).
* Использование осмысленных имен переменных и функций.
* Комментирование сложных участков кода.
* Оптимизацию производительности
* Выбор эффективных алгоритмов и структур данных.
* Минимизация накладных расходов (например, избегание лишних запросов к БД).
* Профилирование и устранение "узких мест".
* Тестирование и отладку
* Написание модульных тестов (unit tests).
* Использование интеграционного тестирования.
* Отладка с помощью профайлеров и логов.
* Интеграцию с внешними системами
* Работа с API сторонних сервисов (платежи, соцсети, облака).
* Обработка ошибок и восстановление после сбоев.

Особенности конструирования основных категорий ПО:

* **Системное ПО:** Конструирование системного ПО требует работы на низком уровне, близком к аппаратному обеспечению. Разработчики используют языки вроде C, C++ или Rust, чтобы обеспечить максимальную производительность и контроль над ресурсами. Ключевые задачи включают управление памятью, обработку прерываний и многопоточную синхронизацию. Поскольку такие программы часто работают в критически важных системах (например, операционные системы или драйверы), особое внимание уделяется отказоустойчивости и безопасности.
* **Прикладное ПО:** При разработке прикладного ПО на первый план выходят удобство интерфейса и функциональность. Языки высокого уровня (Python, Java, C#) ускоряют процесс кодирования, а фреймворки (Qt, .NET, Electron) помогают создавать кроссплатформенные решения. Важно учитывать взаимодействие с пользователем, поэтому тестирование юзабилити играет большую роль. Также часто требуется интеграция с внешними сервисами, такими как базы данных, облачные хранилища или платежные системы.
* **Веб-приложения**: Конструирование веб-приложений подразумевает разделение на клиентскую (frontend) и серверную (backend) части. Frontend строится на HTML, CSS и JavaScript (React, Vue, Angular), а backend — на Node.js, Python (Django, Flask), PHP или Java. Особое внимание уделяется безопасности: защите от SQL-инъекций, XSS-атак и CSRF. Также важна оптимизация загрузки данных, кэширование и работа с API. Современные подходы включают использование микросервисной архитектуры и контейнеризации (Docker, Kubernetes).
* **Мобильные приложения**: Разработка мобильных приложений требует учета особенностей платформ (iOS и Android). Нативные решения пишутся на Swift/Kotlin, а кроссплатформенные — на Flutter или React Native. Ограничения мобильных устройств (батарея, память, производительность) вынуждают оптимизировать код и минимизировать энергопотребление. Также важно тестирование на разных моделях устройств и адаптация под различные размеры экранов.
* **Встраиваемое ПО:** Встраиваемые системы работают в условиях жестких ограничений по памяти, энергопотреблению и вычислительной мощности. Языки типа C и Ada позволяют эффективно управлять ресурсами, а код часто пишется с учетом реального времени (RTOS). Тестирование проводится на реальном оборудовании, так как эмуляторы не всегда точно воспроизводят поведение устройства.

**2. Дайте определение БД, а также определение иерархической, сетевой, реляционной, объектно-ориентированной, объектно-реляционной БД.**

База данных (БД) - совокупность взаимосвязанных данных, организованных в соответствии со схемой данных таким образом, чтобы с ними можно было поддержать эффективную работу конечного пользователя. Проектирование и конструирование любой программной системы, которая предполагает работу с БД, начинаются с проектирования и конструирования структуры данных.

Иерархическая база данных может быть представлена как дерево, состоящее из объектов различных уровней. Самый верхний уровень (корень) занимает один объект, далее идут объекты второго уровня и т.д. Между объектами существуют определенные связи.

Сетевая база данных строится на логической модели данных, которая является расширением иерархического подхода и основана на строгой математической теории, описывающей структурный аспект БД, аспект целостности данных и аспект обработки данных.

Реляционная база данных использует логическую реляционную (табличную) модель данных, прикладную теорию построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и реляционная алгебра.

Объектноориентированная база данных – база данных, в которой данные моделируются в виде объектов, их атрибутов, методов и классов.

Объектно-реляционная база данных – реляционная база данных, поддерживающая некоторые технологии, реализующие объектно-ориентированный подход.

**3. В чем отличие «толстого» клиента от «тонкого»?**

Толстый и тонкий клиент отличаются тем, где происходит основная обработка данных – на стороне клиента или сервера.

Толстый клиент выполняет большую часть обработки данных непосредственно на компьютере пользователя. Это означает, что клиентское приложение содержит значительную часть бизнес-логики и интерфейса, а сервер, как правило, предоставляет только данные. Такой подход требует установки программного обеспечения на каждое клиентское устройство и предъявляет более высокие требования к его техническим характеристикам. Однако толстые клиенты позволяют работать даже при временном отсутствии соединения с сервером, а интерфейс таких приложений, как правило, более отзывчивый и функциональный.

Тонкий клиент, напротив, выполняет минимум операций на стороне пользователя. Основная обработка данных и бизнес-логика сосредоточены на сервере, а клиент используется лишь для отображения интерфейса и передачи команд. Ярким примером тонкого клиента являются веб-приложения, работающие через браузер, или доступ к удалённому рабочему столу через протокол RDP. Такой подход упрощает обновление и поддержку, так как изменения достаточно внести только на сервере. Он также снижает требования к мощности клиентских устройств, поскольку вся тяжёлая обработка происходит на сервере. Однако тонкие клиенты требуют постоянного подключения к сети и не могут полноценно функционировать в оффлайн-режиме.

**4. Какие понятия входят в модель реляционной БД? Перечислите компоненты реляционной модели данных.**

Реляционная база данных – это тип БД, в которой данные организованы в виде связанных таблиц, также называемых отношениями.

В модель реляционной базы данных входят основные понятия, отражающие способ организации и представления данных в виде таблиц. Эти понятия позволяют описывать данные, их взаимосвязи и ограничения целостности.

Основными понятиями в реляционных БД являются:

* Таблицы (отношения) – основной элемент представления реляционной базы данных.
* Строки (кортежи) – это термин, который используется для обозначения строки или записи в таблице. Отдельные экземпляры сущности. Каждая запись в таблице должна быть уникально идентифицирована.
* Столбцы (атрибуты) – столбцы представляют собой отдельные атрибуты данных в таблице. У каждого есть уникальное имя и тип данных, который определяет допустимые значения ячеек.
* Домен – это допустимое множество значений для конкретного атрибута (например, числа, строки, даты).
* Первичные ключи – это один или несколько атрибутов, значения которых уникальны для каждой записи.
* Внешние ключи – это атрибут или их набор в одной таблице, которые ссылаются на первичный ключ или любой другой столбец с уникальными значениями второй таблицы. Это позволяет связывать данные и поддерживать их целостность.

**5. Какие задачи решает проектирование БД?**

Проектирование базы данных – это один из важнейших этапов создания информационной системы. Оно направлено на то, чтобы структура базы данных была логичной, эффективной и соответствовала задачам, которые она должна решать. В процессе проектирования необходимо учитывать несколько ключевых задач.

Первая задача – обеспечение хранения в базе данных всей необходимой информации. Это значит, что в базе должны быть предусмотрены все данные, которые понадобятся пользователям или системе в процессе работы. Для этого проектировщик определяет перечень объектов (сущностей), с которыми будет вестись работа, а также их свойства (атрибуты). Например, для системы учёта заказов сущностями могут быть клиенты, товары и заказы, а атрибутами – имя клиента, наименование товара, количество, дата покупки и так далее. Структура таблиц должна быть продумана так, чтобы все важные данные были зафиксированы, не терялись и были легко доступны.

Вторая задача – обеспечение возможности получения данных по всем необходимым запросам. Важно не только сохранить информацию, но и предоставить к ней удобный и быстрый доступ. Пользователи и приложения должны иметь возможность выполнять выборки, фильтрацию, сортировку, объединение данных из разных таблиц и формировать отчёты. Поэтому при проектировании нужно заранее продумать, какие именно запросы будут использоваться, какие связи между таблицами потребуются и как обеспечить эффективное выполнение этих операций. Также важно учитывать производительность, особенно при работе с большими объёмами данных.

Третья задача – сокращение избыточности и дублирования данных. Повторяющаяся информация увеличивает объём базы и может привести к ошибкам при обновлении. Например, если имя одного и того же клиента записано в разных таблицах и отличается хотя бы одной буквой, это может нарушить целостность данных. Чтобы избежать этого, используется нормализация – процесс разделения данных на связанные таблицы, чтобы каждая единица информации хранилась только в одном месте. Это повышает согласованность и упрощает обслуживание базы.

Четвёртая задача – обеспечение целостности базы данных. Целостность означает, что данные в базе должны оставаться логически непротиворечивыми. Например, нельзя допустить, чтобы в заказе указывался несуществующий товар или чтобы у клиента не было идентификатора. Для этого используются ограничения: первичные и внешние ключи, правила проверки значений (например, значение должно быть больше нуля), запреты на пустые значения в обязательных полях и т. д. Такие механизмы предотвращают появление некорректных или потерянных данных и поддерживают структуру базы в устойчивом состоянии.

**6. Дайте определение концептуального (инфологического) проектирования. Опишите принцип построения ER-диаграммы для концептуальной модели данных.**

Концептуальное (инфологическое) проектирование – построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами. Кроме того, в этом контексте равноправно могут использоваться слова «модель базы данных» и «модель предметной области» (например, «концептуальная модель базы данных» и «концептуальная модель предметной области»), поскольку такая модель является как образом реальности, так и образом проектируемой базы данных для этой реальности.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя:

* описание информационных объектов или понятий предметной области и связей между ними.
* описание ограничений целостности, то есть требований к допустимым значениям данных и к связям между ними.

Проектирование базы данных в контексте построения ER-диаграммы начинается с определения сущностей – объектов, о которых необходимо хранить информацию. Каждая сущность изображается в виде прямоугольника с указанием имени, а под ней записывается ключ – уникальный идентификатор. Далее определяются связи между сущностями, которые обозначаются ромбами с глаголом внутри, отражающим суть отношения между объектами. Важно указать степень связи: один к одному (1:1), один ко многим (1:М или М:1) и многие ко многим (М:М). Одна и та же пара сущностей может иметь несколько разных связей. Каждая сущность также может иметь класс принадлежности: обязательный (О), если каждый экземпляр должен участвовать в связи, и необязательный (Н), если участие не обязательно.

**7. Дайте определение даталогического проектирования. Опишите принципы построения ER-диаграммы для даталогической модели данных.**

Логическое (даталогическое) проектирование – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель – набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. Этот этап может быть в значительной степени автоматизирован.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Для отображения логической структуры данных используется ER-диаграмма (диаграмма «сущность–связь»), которая строится по определённым принципам. Вначале определяются сущности – объекты предметной области, данные о которых необходимо хранить. Каждая сущность представляется в виде прямоугольника с названием и включает в себя атрибуты – свойства сущности, среди которых обязательно выделяется ключевой атрибут (идентификатор). Далее между сущностями устанавливаются связи, обозначаемые ромбами с глагольным названием, отражающим характер взаимодействия. К связи присоединяются сущности с указанием степени связи: один к одному (1:1), один ко многим (1:М или М:1) и многие ко многим (М:М). Также указывается класс принадлежности: обязательный (О), если каждый экземпляр сущности должен участвовать в связи, или необязательный (Н), если участие не обязательно.

Таким образом, ER-диаграмма в даталогическом проектировании служит графическим представлением логической структуры базы данных, помогает выявить и устранить избыточность, обеспечивает целостность и взаимосвязь данных, а также создаёт основу для последующего перехода к физической модели.

**8. Дайте определение физического проектирования БД.**

Физическое проектирование – создание схемы базы данных (рис. 1) для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектов базы данных, ограничения на поддерживаемые типы данных и т. п. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и т. д.

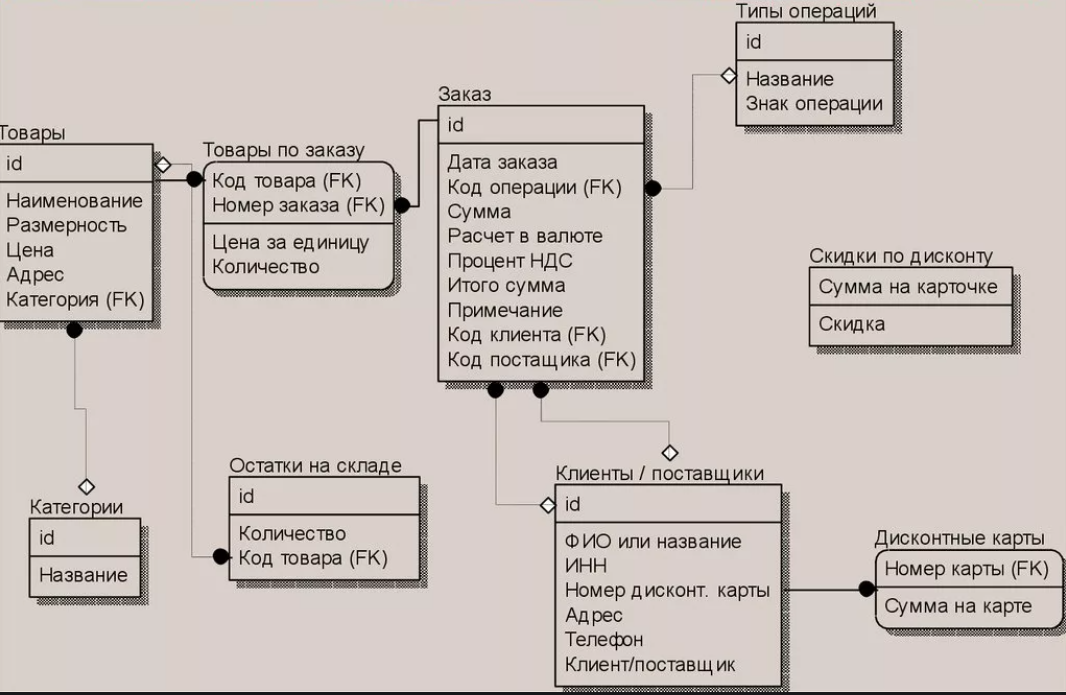


Рис. 1 Схема базы данных

Этап 1. Проектирование основных отношений.

Определение каждого отношения включает:

* имя отношения;
* список простых атрибутов;
* определение первичного ключа, альтернативных (АК) и внешних (ВК) ключей;
* список производных атрибутов и описание способов их вычисления;
* определение требований ссылочной целостности для любых внешних ключей.

Для каждого атрибута следует указать:

* определение его домена, включающее тип данных, размерность внетреннего представления атрибута и любые требуемые ограничения на допустимые значения;
* значение атрибута, принимаемое по умолчанию;
* допустимое значение NULL.

Этап 2. Разработка способов получения производных данных.

Производные или расчетные атрибуты – атрибуты, значения которых можно определить, используя значения других атрибутов.

Производный атрибут может храниться в БД или вычисляться каждый раз, когда в нем возникает необходимость.

Два варианта затрат:

* на хранение производных данных и поддержание их согласованности с реальными данными
* на вычисление производных данных по мере необходимости

Этап 3. Реализация ограничений предметной области.

Способы реализации ограничений:

* с использованием возможностей стандарта языка SQL
* с применением триггеров
* непосредственно в самом приложении

**9. Каковы особенности конструирования логики работы с данными для СУБД?**

Под конструированием логики работы с данными понимается разработка процедур и функций, которые позволяют осуществлять как обработку, так и контроль хранимой информации средствами СУБД.

Простейшие системы, которые создаются с использованием баз данных, ограничивают разработку баз данных созданием структуры таблиц и связей между ними. Вся обработка данных осуществляется из приложения, которое предоставляет и обрабатывает информацию, а затем сохраняет ее в базе данных. Данный подход является наименее безопасным, так как любой пользователь, получивший логин и пароль к базе данных, используя приложение, получает доступ к таблицам с данными и может не только считывать информацию, но и изменять ее. Такой тип приложений называется «с толстым клиентом».

Более безопасным считается подход, когда доступ к таблицам данных имеют только процедуры СУБД, которые вызываются приложением. Это позволяет в самих процедурах контролировать целостность данных и не выполнять некорректные действия. При этом доступ приложения непосредственно к таблицам полностью запрещается. В данном случае необходимо для каждой таблицы создавать как минимум четыре процедуры: процедура на чтение данных, процедура добавления данных, процедура изменения данных и процедура удаления данных.

Подход к работе с данными через процедуры требует программирования на уровне баз данных. Дальнейшим развитием этого подхода является программирование всей логики обработки данных в БД, а приложение должно только отображать информацию, осуществлять первичную проверку корректности вводимых данных и вызывать соответствующие процедуры в СУБД. Такой тип приложений называется «с тонким клиентом».

Существуют решения, которые выносят часть логики обработки информации (так называемая бизнес-логика) на отдельный сервер приложений. Клиентские приложения обращаются к серверу приложений, который обрабатывает запросы и вызывает соответствующие процедуры базы данных. Данный подход является наиболее безопасным, так как становится практически невозможным получить доступ к БД.

**10. Расскажите о безопасности БД. Каковы основные варианты обеспечения защиты данных?**

Вопросы безопасности баз данных тесно связаны с защитой хранимой информации. Чтобы информация не попала к злоумышленникам, организациям необходимо применять комплексные меры по защите баз данных. Вот основные варианты обеспечения безопасности данных в СУБД:

1. Безопасность на уровне прав доступа. Все современные СУБД поддерживают возможность настройки прав доступа практически на любой объект базы данных, включая таблицы, представления, процедуры и другие элементы. Каждое приложение или пользователь, подключающийся к базе данных, должен пройти процедуру авторизации, которая проверяет его права и разрешает или ограничивает доступ в зависимости от назначенных прав. Это предотвращает несанкционированный доступ и модификацию данных.

2. Совмещенная безопасность на уровне архитектуры БД и прав доступа. Этот подход предполагает запрет прямого доступа к таблицам базы данных для приложений. Вместо этого, все запросы к данным должны быть обработаны через специально разработанные процедуры в СУБД. Приложения имеют доступ только к этим процедурам, которые могут выполнять проверки целостности данных и обеспечивать корректное их изменение. Это значительно повышает безопасность, так как предотвращает возможность выполнения некорректных или вредоносных операций с базой данных.

3. Защищенные соединения. Важно, чтобы все подключения к базе данных осуществлялись по защищенному каналу. Наиболее распространённым методом защиты соединений является использование протокола SSL (Secure Sockets Layer), который обеспечивает шифрование данных, передаваемых между клиентом и сервером. Это защищает данные от перехвата или подделки в процессе их передачи по сети.

4. Шифрование данных. Даже если соединение защищено с использованием SSL, существует риск, что злоумышленник может получить доступ к самой базе данных, например, через уязвимости в приложении или неправильные настройки безопасности. В таких случаях шифрование данных на уровне базы данных является важным методом защиты. Шифрование данных гарантирует, что даже в случае утечки информации, она будет оставаться непригодной для использования без соответствующего ключа дешифрования.

5. Обеспечение безопасности при разработке. Очень важно учитывать безопасность на стадии разработки. Ошибки в программировании могут привести к уязвимостям, таким как SQL-инъекции. Это происходит, когда запросы к базе данных формируются динамически, и злоумышленник может изменить или подставить вредоносный код в строку запроса, что приведет к несанкционированному доступу к данным. Чтобы предотвратить это, необходимо использовать подготовленные запросы и другие методы защиты от SQL-инъекций, такие как экранирование входных данных.

6. Мониторинг и аудит безопасности. Для своевременного обнаружения попыток взлома или неправомерного доступа важно внедрить системы мониторинга и аудита. Эти системы отслеживают все действия, происходящие в базе данных, и записывают информацию о пользователях, их запросах и любых изменениях в данных. Это позволяет оперативно выявить подозрительную активность и быстро реагировать на возможные угрозы.

7. Резервное копирование и восстановление данных. Еще одной важной частью безопасности является регулярное создание резервных копий данных и разработка стратегии восстановления в случае потери данных. Эти меры обеспечивают защиту информации от потерь, вызванных не только атаками злоумышленников, но и случайными сбоями, ошибками пользователей или аппаратными проблемами.

**11. Что такое SQL-инъекции, как можно защитить от них приложение?**

SQL-инъекции – это тип уязвимости в приложениях, взаимодействующих с базами данных, когда злоумышленник может вставить или «впрыснуть» вредоносный SQL-код в запрос, который выполняет приложение. Это позволяет ему манипулировать SQL-запросами, отправляемыми в базу данных, и получать несанкционированный доступ к данным, их изменять или удалять.

Типичный пример SQL-инъекции – это приложение, которое принимает входные данные от пользователя и использует их для формирования SQL-запроса, например, для авторизации пользователя. Если запрос строится с учётом пользовательских данных без должной проверки, злоумышленник может ввести в поле логина или пароля специальный SQL-код, который приведёт к изменению логики запроса и получению доступа к базе данных.

Чтобы защититься от SQL-инъекций, нужно использовать подготовленные запросы, которые отделяют данные от SQL-кода, предотвращая их интерпретацию как команды. Также полезно применять хранимые процедуры, так как они заранее подготавливают запросы и принимают параметры, что исключает возможность инъекции.

Кроме того, необходимо экранировать пользовательские данные, обрабатывая специальные символы, такие как ', -- и ;, чтобы они не влияли на запросы. Ограничение прав доступа приложений и валидация входных данных также помогают уменьшить риски.

**12. Каковы основные преимущества структурного программирования?**

Структурное программирование представляет собой подход, который включает определенные принципы и правила разработки программ с целью упрощения их создания и тестирования, повышения производительности программистов и улучшения читаемости кода. Основным принципом структурного программирования является отказ от беспорядочного стиля кодирования, в частности, от использования оператора Goto. Применение этих принципов позволяет существенно повысить производительность труда программистов, в некоторых случаях в 5-6 раз.

Основой структурного программирования является теорема о структурировании, которая утверждает, что любой алгоритм для решения вычислимой задачи можно представить с использованием трех базовых управляющих структур: последовательности, ветвления и цикла. Эти структуры являются основными элементами, из которых строится программа. Структура следования обеспечивает выполнение операторов один за другим, структура ветвления позволяет принимать решения в зависимости от условия, а структура цикла позволяет повторять действия до выполнения определенного условия.

Основные преимущества структурного программирования заключаются в упрощении разработки и тестирования программ. Этот подход требует использования четкой структуры, что позволяет легче писать, отлаживать и тестировать код. Программы становятся более логичными и понятными, что упрощает их чтение и понимание другими разработчиками. Структурное программирование также способствует улучшению читабельности кода за счет отказа от оператора Goto, что делает программу более предсказуемой и менее подверженной ошибкам.

Кроме того, благодаря использованию базовых управляющих структур, таких как последовательность, ветвление и цикл, код становится более ясным и логичным. Это значительно увеличивает производительность труда программистов, так как они могут сосредоточиться на решении задачи, а не на технических аспектах реализации. Также структурное программирование поддерживает модульность, позволяя разбивать программу на подпрограммы и модули, которые могут быть использованы повторно в других проектах. В результате создается более организованный и гибкий код, который легче поддерживать и развивать.

**13. Перечислите основные конструкции структурного программирования. В чем их особенность? Расскажите о методике разработки программ использованием структурного программирования.**

Основными конструкциями структурного программирования являются структуры следования, ветвления и цикла. Эти конструкции составляют основу любой программы, обеспечивая четкую логику выполнения кода. Структура следования представляет собой последовательность операторов, которые выполняются один за другим. Она является самой простой конструкцией, которая отражает естественный ход выполнения программы. Структура ветвления используется для принятия решений в программе. Она включает проверку некоторого логического условия, и в зависимости от его выполнения, выполняются разные части кода. Обычно это реализуется с помощью оператора if-else. Структура цикла же отвечает за многократное выполнение одного и того же блока кода при условии, что заданное логическое условие выполняется. В языках программирования эта конструкция часто реализуется с помощью операторов, таких как while или for.

Особенность этих конструкций заключается в том, что каждая из них имеет один вход и один выход, что делает программы более структурированными и понятными. Они могут быть вложенными, но в конечном итоге вся программа состоит из этих простых блоков, что значительно упрощает как понимание программы, так и ее отладку.

Методика разработки программ с использованием структурного программирования обычно основывается на подходе "сверху вниз". Этот метод начинается с определения целей и задач программы, после чего идет постепенная детализация. Сначала разрабатываются самые общие модули, которые затем уточняются и разбиваются на более мелкие части. Такой подход позволяет ускорить процесс разработки, поскольку разработчики могут сосредоточиться на выполнении более крупных задач, откладывая решение второстепенных проблем. Важно отметить, что в процессе разработки программа должна быть разделена на модули и подпрограммы, каждая из которых выполняет одну конкретную задачу. Это позволяет не только улучшить организацию кода, но и обеспечивать его повторное использование в других частях программы.

**14. Дайте определение объектно-ориентированного программирования. Какие основные понятия положены в основу объектно-ориентированного программирования?**

Объектно-ориентированное программирование (ООП) – это парадигма программирования, в основе которой лежат концепции объектов и классов. В этой парадигме программы создаются как коллекции взаимодействующих объектов, каждый из которых имеет свое состояние и поведение. ООП ориентировано на представление данных в виде объектов, что позволяет моделировать реальный мир и взаимодействие объектов между собой. Программы, построенные с использованием ООП, включают описание классов, которые являются шаблонами для создания объектов.

Основными понятиями ООП являются абстракция данных, класс, интерфейс класса, прототип, абстрактный класс и объект. Абстракция данных позволяет скрывать детали реализации и предоставлять лишь необходимые интерфейсы для работы с данными. Класс описывает структуру объектов и поведение, которое эти объекты могут демонстрировать. Каждый класс включает в себя поля (состояние) и методы (поведение), которые могут быть публичными (доступными для других частей программы), защищенными (доступными только в рамках класса и его наследников) или закрытыми (доступными только внутри самого класса). Интерфейс класса представляет собой набор публичных методов и полей, через которые объект класса взаимодействует с внешней средой.

Прототип – это объект-образец, по которому создаются другие объекты. Абстрактный класс – это класс, который содержит абстрактные методы, не реализованные в нем, но обязательные для реализации в классах-наследниках. Объект, в свою очередь, является основной единицей ООП, обладающей состоянием и поведением, которые определяются классом. Важным моментом является инстанцирование – процесс создания экземпляра класса, который является конкретным объектом, существующим в памяти.

**15. Какими тезисами можно кратко выразить концепции объектно-ориентированного программирования?**

Концепции объектно-ориентированного программирования можно выразить следующим образом. В основе ООП лежит понятие объекта, который представляет собой сущность с состоянием и поведением. Объекты создаются на основе классов, которые задают структуру и методы, доступные для этих объектов. Одной из ключевых концепций является абстракция, которая позволяет скрывать детали реализации и предоставлять только необходимые интерфейсы для взаимодействия с объектом.

Инкапсуляция обеспечивает защиту данных объекта, скрывая внутреннюю реализацию и предоставляя доступ к данным и методам только через публичные интерфейсы. Наследование позволяет создавать новые классы на основе существующих, что способствует повторному использованию кода и расширению функциональности. Полиморфизм, в свою очередь, позволяет использовать объекты разных типов через общий интерфейс, вызывая методы, которые могут вести себя по-разному в зависимости от типа объекта.

Прототипы и абстрактные классы играют роль в создании объектов-образцов, которые могут быть использованы для создания новых объектов, а также в определении общих характеристик для классов, которые не реализуют всю функциональность, но служат основой для наследников. Процесс инстанцирования заключается в создании экземпляра класса, который становится конкретным объектом в программе, готовым к использованию.

**16. Дайте определение наследования, инкапсуляции, полиморфизма. Поясните смысл их использования.**

Наследование, инкапсуляция и полиморфизм являются основными концепциями объектно-ориентированного программирования (ООП), каждая из которых выполняет свою уникальную роль и помогает эффективно решать задачи при проектировании и разработке программ.

Наследование позволяет создавать новый класс на основе уже существующего, называемого родительским или базовым классом. Это один из способов повторного использования кода, который существенно упрощает разработку, сокращая необходимость переписывать одну и ту же логику в разных частях программы. Класс-наследник получает все свойства и методы родительского класса, что позволяет использовать их без изменений. Однако, наследник может переопределить или расширить функциональность родительского класса, добавив собственные методы или изменив существующие. Наследование способствует иерархическому структурированию классов, что позволяет описывать более сложные объекты, начиная от простых базовых и заканчивая специализированными производными классами.

Инкапсуляция связана с объединением данных и методов, работающих с ними, в один объект, и с сокрытием деталей реализации этого объекта от внешнего мира. Это позволяет пользователю работать с объектом на более высоком уровне абстракции, не заботясь о том, как именно реализована его функциональность. Инкапсуляция также повышает безопасность данных, поскольку она ограничивает доступ к внутренним данным объекта. Для этого используются модификаторы доступа, такие как private (закрытый доступ), protected (защищённый доступ) и public (открытый доступ). Например, если у объекта есть свойство "баланс", то оно может быть доступно только через специальные методы, такие как "положить деньги" или "снять деньги", предотвращая прямое изменение баланса извне, что может нарушить логику работы программы. Это обеспечивает более безопасную и контролируемую работу с данными и уменьшает вероятность возникновения ошибок.

Полиморфизм позволяет объектам с одинаковым интерфейсом вести себя по-разному, что дает возможность использовать объекты разных типов через общий интерфейс. В ООП полиморфизм реализуется с помощью наследования и интерфейсов. Суть полиморфизма в том, что методы одного и того же имени могут иметь разные реализации в разных классах. Например, класс "Животное" может иметь метод "звуки", который будет по-разному реализован в классах "Собака" и "Кошка". В случае полиморфизма можно вызвать метод "звуки" для объекта любого типа "Животное", и он выполнит корректную реализацию в зависимости от типа объекта. Это существенно упрощает работу с объектами, так как позволяет обрабатывать их через единый интерфейс, не беспокоясь о том, с каким конкретно классом мы работаем. Полиморфизм делает код более гибким, расширяемым и удобным для работы с различными типами объектов без необходимости переписывать логику.

**17. Дайте определение конструктора класса. Какие основные виды конструкторов бывают? Дайте определение деструктора.**

Конструктор в объектно-ориентированном программировании (ООП) – это специальный метод, предназначенный для инициализации объекта при его создании. Конструктор выполняет важную задачу: он отвечает за подготовку объекта к работе, выделение памяти и инициализацию полей или свойств объекта значениями. В разных языках программирования конструктор имеет свои особенности, но основная задача остаётся неизменной. В языках C++, C# или Java конструктор класса имеет то же имя, что и сам класс. Он вызывается автоматически, когда создаётся новый объект, и не может быть вызван напрямую, как обычный метод.

Существуют различные виды конструкторов, каждый из которых служит определённой цели:

1. Конструктор по умолчанию – это конструктор, который не имеет обязательных аргументов. Он вызывается автоматически, если при создании объекта не передаются параметры. В случае, если разработчик не определил явным образом конструктор по умолчанию, компилятор автоматически генерирует его сам, инициализируя поля объекта значениями по умолчанию. Например, если в классе есть целочисленное поле, конструктор по умолчанию может установить его значение равным 0.

2. Конструктор копирования – это конструктор, принимающий в качестве аргумента ссылку на объект того же класса. Он используется для создания нового объекта, который является копией другого. Такой конструктор копирует значения полей из исходного объекта в новый. Это особенно важно при работе с динамически выделенной памятью или объектами, требующими специфической логики копирования. В случае с динамическим выделением памяти, конструктор копирования может быть использован для корректного копирования указателей на память, чтобы избежать ошибок, таких как утечка памяти.

3. Обычный конструктор – это конструктор, который имеет произвольное количество параметров, и используется для инициализации объекта с заданными значениями. В отличие от конструктора по умолчанию, обычный конструктор требует передачи аргументов при создании объекта. Например, если класс "Треугольник" имеет два поля (основание и высоту), конструктор этого класса может принимать два параметра – значение основания и высоты, чтобы правильно инициализировать объект.

Деструктор – это метод, который служит для деинициализации объекта и освобождения всех ресурсов, которые были выделены для него. Например, деструктор может освободить память, закрыть файлы или сетевые соединения, которые были открыты в процессе работы объекта. Имя деструктора совпадает с именем класса, но перед ним стоит префикс ~. Деструкторы автоматически вызываются, когда объект выходит за пределы области видимости или когда он явно уничтожается, и задача деструктора — корректно очистить все ресурсы, используемые объектом, чтобы избежать утечек памяти и других проблем.

**18. Дайте определение виртуальной функции. Чем переопределение виртуальной функции в дочернем классе отличается от переопределения невиртуальной?**

Виртуальная функция – это функция-член класса, которая может быть переопределена в производном (дочернем) классе и позволяет обеспечить полиморфизм на уровне времени выполнения. Объявляется она с помощью ключевого слова virtual в базовом (родительском) классе. При вызове виртуальной функции через указатель или ссылку на базовый класс будет вызвана реализация функции из дочернего класса, если она переопределена, даже несмотря на то, что переменная имеет тип базового класса.

Переопределение виртуальной функции в дочернем классе отличается от переопределения невиртуальной тем, что:

* При виртуальной функции выбор, какую версию функции вызывать (из базового или дочернего класса), происходит во время выполнения программы. Это и есть динамический полиморфизм, позволяющий писать гибкий и расширяемый код, особенно при использовании наследования.
* При невиртуальной функции выбор версии функции происходит на этапе компиляции (то есть заранее, статически), исходя из типа указателя или ссылки, а не от фактического типа объекта, на который он указывает. Это значит, что при вызове через указатель на базовый класс будет вызвана функция базового класса, даже если в дочернем классе определена своя реализация с таким же именем.

**19. Дайте определение шаблона проектирования. Как в общем виде описываются шаблоны? Опишите принципы работы с шаблонами.**

Шаблон проектирования – это проверенное, эффективное решение типовой задачи, которая часто встречается в процессе разработки программного обеспечения. Он представляет собой своего рода универсальную модель решения, которую можно адаптировать под конкретную ситуацию при проектировании или построении архитектуры приложения.

В основе каждого шаблона лежит описание типичной проблемы, с которой разработчики сталкиваются в различных контекстах, а также способ её решения. Главная цель шаблонов – не просто предоставить готовый код, а предложить концептуальный подход, который можно повторно использовать в разных проектах. Они помогают повысить уровень абстракции и стандартизировать процесс разработки.

Каждый шаблон описывается с использованием четырех основных компонентов. Имя шаблона даёт возможность кратко и ясно сослаться на идею, которую он воплощает, без необходимости в каждый раз объяснять детали. Задача объясняет, в каких ситуациях уместно применять данный шаблон, включая конкретные проблемы и контексты, в которых он эффективен. Решение содержит архитектурное описание – какие классы или объекты используются, как они связаны друг с другом, какие обязанности несут и как взаимодействуют. Это не конкретная реализация, а обобщённая структура, которая может быть адаптирована под разные случаи. Наконец, результаты отражают последствия применения шаблона – каковы его преимущества, какие компромиссы нужно учесть, и как он влияет на гибкость, расширяемость и повторное использование кода.

Работа с шаблонами проектирования требует осознанного подхода. Сначала необходимо ознакомиться с описанием шаблона, чтобы понять его назначение и логику. Затем важно разобраться во всех классах и объектах, упомянутых в шаблоне, и в том, как они взаимодействуют. Когда шаблон адаптируется для конкретной задачи, часто требуется придумать собственные имена для его участников, поскольку исходные термины могут быть слишком абстрактными. После этого нужно создать необходимые классы, определить их интерфейсы и связи между ними, чтобы воплотить структуру шаблона в коде. Последним шагом является определение операций, которые реализуют взаимодействие между элементами шаблона в соответствии с требованиями конкретного приложения.

**20. Опишите типы шаблонов проектирования, чем они отличаются и для чего служат.**

**1. Порождающие шаблоны (Creational Patterns)**

Эти шаблоны отвечают за создание объектов, обеспечивая гибкость и удобство управления процессом инстанцирования.

Абстрактная фабрика: Создает семейства взаимосвязанных объектов без указания их конкретных классов.

Одиночка: Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Прототип: Позволяет копировать объекты, не вдаваясь в детали их реализации.

Строитель: Позволяет создавать сложные объекты пошагово, используя один и тот же процесс конструирования.

Фабричный метод: Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.

**2. Структурные шаблоны (Structural Patterns).**

Эти шаблоны помогают организовать классы и объекты в более крупные структуры, сохраняя гибкость и эффективность.

Адаптер: Преобразует интерфейс одного класса в интерфейс, ожидаемый клиентом.

Декоратор: Динамически добавляет объекту новые обязанности, не меняя его исходного класса.

Заместитель: Предоставляет суррогат или заместитель для другого объекта, чтобы контролировать доступ к нему.

Компоновщик: Объединяет объекты в древовидные структуры, позволяя клиентам работать с ними единообразно.

Мост: Разделяет абстракцию от реализации, позволяя им изменяться независимо.

Приспособленец: Экономит память, разделяя общее состояние между множеством объектов.

Фасад: Предоставляет упрощенный интерфейс к сложной подсистеме.

**3. Поведенческие шаблоны (Behavioral Patterns).**

Эти шаблоны определяют способы взаимодействия объектов и распределения обязанностей между ними.

Интерпретатор: Определяет грамматику языка и интерпретирует предложения этого языка.

Итератор: Предоставляет способ последовательного доступа к элементам агрегированного объекта.

Команда: Инкапсулирует запрос в виде объекта, позволяя параметризировать клиентов с различными запросами.

Наблюдатель: Определяет зависимость "один-ко-многим" между объектами, чтобы при изменении состояния одного объекта все зависящие от него объекты автоматически уведомлялись.

Посетитель: Позволяет добавлять новые операции к объектам без изменения их классов.

Посредник: Упрощает взаимодействие между объектами, инкапсулируя их коммуникацию в отдельный объект.

Состояние: Позволяет объекту изменять свое поведение при изменении внутреннего состояния.

Стратегия: Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

Хранитель: Позволяет сохранять и восстанавливать предыдущее состояние объекта.

Цепочка обязанностей: Позволяет передавать запросы последовательно по цепочке обработчиков.

Шаблонный метод: Определяет скелет алгоритма, позволяя подклассам переопределять некоторые его шаги.

**21. Дайте определение системы управления версиями. Зачем они нужны?**

Система управления версиями (англ. Version Control System, VCS) – это специализированное программное обеспечение, предназначенное для эффективной работы с часто изменяющейся информацией, такой как исходный код программ. Она позволяет отслеживать все изменения, которые происходят с файлами проекта во времени, а также управлять совместной работой нескольких разработчиков над одним и тем же проектом.

Главная задача системы управления версиями – обеспечить сохранность истории изменений и дать возможность вернуться к предыдущим состояниям проекта в любой момент. Благодаря этому разработчики могут без опасений экспериментировать, зная, что всегда смогут восстановить рабочую версию.

Системы управления версиями предоставляют целый ряд полезных возможностей. Они позволяют создавать ветки – независимые варианты одного и того же проекта, которые развиваются параллельно. При этом вся история изменений до момента ветвления остаётся общей. Также можно определить, кто и когда изменил конкретные строки в файле, что особенно важно при командной разработке. Все изменения сопровождаются комментариями (журналом изменений), где разработчик указывает, что именно он сделал и зачем. Кроме того, система может контролировать права доступа, ограничивая чтение или изменение файлов для разных пользователей.

Работа с системой управления версиями начинается с извлечения – разработчик получает рабочую копию проекта на свой компьютер. Далее, в ходе обычного рабочего цикла, он выполняет обновление своей копии, чтобы синхронизироваться с последними изменениями в основном проекте. Затем он вносит собственные изменения локально, не обращаясь к серверу, и, завершив задачу, фиксирует изменения, передавая их обратно в систему.