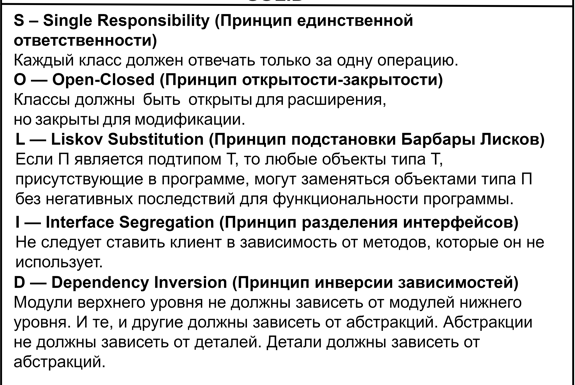
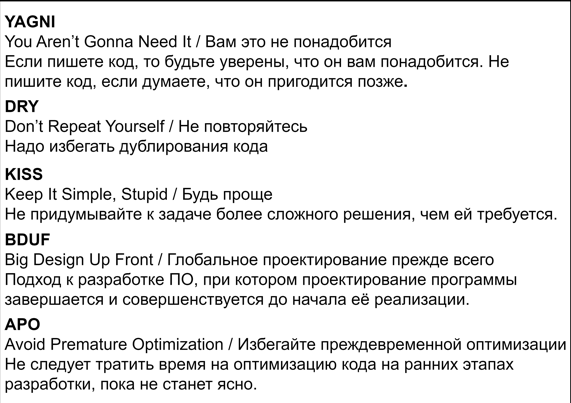
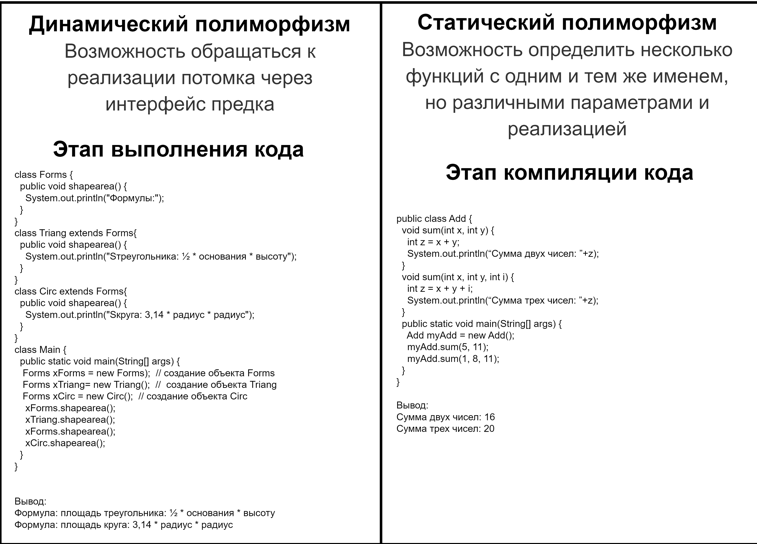
1. **SOLID **
2. **KISS, DRY, YAGNI, BDUF, APO **
3. **Что такое объектно-ориентированное программирование (ООП)?** ООП — это парадигма программирования, основанная на концепции объектов, которые могут содержать данные (атрибуты) и методы (функции). Программы строятся вокруг взаимодействия этих объектов.

1. **Каковы основные принципы ООП?**

**Инкапсуляция** — сокрытие данных объекта и предоставление доступа к ним через методы; механизм, позволяющий объединить данные и методы, работающие с этими данными в единый объект и скрыть детали реализации от пользователя.

**Наследование** — создание новых классов на основе существующих с добавлением или изменением функционала.

**Полиморфизм** — возможность работы с объектами разных типов через единый интерфейс; механизм ООП, который позволяет разным сущностям выполнять одни и те же действия. При этом нне важно, как эти сущности устроены внутри и чем они различаются.



**Абстрагирование** — выделение значимых характеристик объекта и игнорирование незначимых. Выделение общих характеристик объектов, их свойств и методов, при игнорировании деталей реализации.

1. **Объясните концепцию инкапсуляции в ООП**. **Инкапсуляция** — это механизм, при котором данные и методы, работающие с ними, объединяются в класс, а доступ к ним контролируется через модификаторы доступа (например, private, public).

Принцип независимости данных в ООП, т.е. можно скрыть некие данные/функционал от др пользователей

Инкапсуляцию также описывают как принцип разделения логики и поведения. Логика — то, как что-то устроено внутри. Поведение — то, как оно взаимодействует с другими сущностями. Разделение этих двух понятий упрощает взаимодействие объектов в коде. Обычно содержимое заключается в специальную программную оболочку, которая закрывает данные от внешних обращений.

Поэтому инкапсуляция нужна — так код становится более упорядоченным и повышается отказоустойчивость. Ведь если сломается инкапсулированный объект, это не повлияет на работу других. Примерно как сломанная кнопка не приводит к поломке всего устройства.

1. **Как наследование способствует повторному использованию кода в ООП? Наследование** позволяет повторно использовать код, создавая новый класс (подкласс), который наследует свойства и методы существующего класса (суперкласса). **Наследование в ооп —** это концепция, согласно которой одни классы, называемые родительскими, могут лежать в основе других — дочерних. При этом, дочерние классы перенимают свойства и поведение своего родителя и должны их как-то дополнять.

При создании объекта сначала создается конструктор родителя, затем потомка.

1. **Какова цель полиморфизма в ООП?** Полиморфизм – возможность объекта вести себя по-разному. Полиморфизм позволяет одному интерфейсу работать с объектами разных типов. Это упрощает расширяемость и гибкость кода.

Например, есть две разных сущности: картинка и видео. И тем, и другим можно поделиться: отправить в личное сообщение другому человеку. Программист может сделать два разных метода — один для картинки, другой для видео. А может воспользоваться полиморфизмом и создать один метод «Отправить» для обеих сущностей. Такой метод будет называться полиморфным. Плюс этого подхода — разработчик пишет меньше кода и не повторяется.

1. **Опишите разницу между классом и объектом. Класс** — это «шаблон» для объекта, который описывает его свойства. Несколько похожих между собой объектов, например профили разных пользователей, будут иметь одинаковую структуру, а значит, принадлежать к одному классу. Каждый объект — это экземпляр какого-нибудь класса.
2. **Что такое конструктор в ООП и какова его роль? constructor**— это специальный метод, который автоматически вызывается при создании объекта. Конструктор может принимать любые аргументы, как и любой другой метод. В каждом языке конструктор обозначается своим именем. Где-то это специально зарезервированные имена типа \_\_construct или \_\_init\_\_, а где-то имя конструктора должно совпадать с именем класса. Назначение конструкторов — произвести первоначальную инициализацию объекта, заполнить нужные поля.Не дает инициализировать объект мусором. Если не прописываем, то компилятор делает все сам (конструктор по умолчанию).В отличие от сеттера вызывается сам при создании класса, размещ только в паблик секции.
3. **Объясните концепцию абстракции в ООП. Абстракция** — это выделение ключевых характеристик объекта, игнорируя его менее важные детали. Она помогает управлять сложностью путем сокрытия ненужных деталей реализации. Абстрагирование — это способ выделить набор наиболее важных атрибутов и методов и исключить незначимые. Соответственно, абстракция — это использование всех таких характеристик для описания объекта. Важно представить объект минимальным набором полей и методов без ущерба для решаемой задачи.
4. **Как классы и объекты связаны с объектами реального мира?**

Объектно-ориентированная идеология разрабатывалась как попытка связать поведение сущности с её данными и спроецировать объекты реального мира и бизнес-процессов в программный код. Задумывалось, что такой код проще читать и понимать человеком, т. к. людям свойственно воспринимать окружающий мир как множество взаимодействующих между собой объектов, поддающихся определенной классификации.

1. **Каковы преимущества использования ООП перед процедурным программированием?** До ООП в разработке использовался другой подход — процедурный. Программа представляется в нем как набор процедур и функций — подпрограмм, которые выполняют определенный блок кода с нужными входящими данными. Процедурное программирование хорошо подходит для легких программ без сложной структуры. Но если блоки кода большие, а функций сотни, придется редактировать каждую из них, продумывать новую логику. В результате может образоваться много плохо читаемого, перемешанного кода — «спагетти-кода» или «лапши».

В отличие от процедурного, объектно-ориентированное программирование позволяет вносить изменения один раз — в объект. Именно он — ключевой элемент программы. Все операции представляются как взаимодействие между объектами. При этом код более читаемый и понятный, программа проще масштабируется

 Улучшенная модульность кода. Повторное использование кода через наследование. Более лёгкое сопровождение и тестирование благодаря инкапсуляции. Гибкость и расширяемость кода за счет полиморфизма.

1. **Определите класс и приведите пример.**
2. **Что такое объект и чем он отличается от класса? Класс** — это описание того, какими свойствами и поведением будет обладать объект. А **объект** — это экземпляр с собственным состоянием этих свойств.
3. **Как создать объект из класса в ООП?**
4. **Указатель this this**— это специальная локальная переменная (внутри методов), которая позволяет объекту обращаться из своих методов к собственным атрибутам. Обращаю внимание, что только к собственным, то бишь, когда трансформер вызывает свой метод, либо меняет собственное состояние. Если снаружи обращение будет выглядеть так: **optimus.x**, то изнутри, если Оптимус захочет сам обратиться к своему полю x, в его методе обращение будет звучать так: **this->x**, то есть "*я (Оптимус) обращаюсь к своему атрибуту x*". В большинстве языков эта переменная называется this, но встречаются и исключения (например, self)Если не пишем, то компилятор использует неявно. Это указатель объекта на самого себя
5. **Как ООП помогает моделировать объекты реального мира с помощью классов? ООП моделирует объекты реального мира**, предоставляя классы, которые описывают их поведение и характеристики.
6. **Каково значение ключевого слова «new» при создании объектов? new**— это ключевое слово, которое необходимо использовать для создания нового экземпляра какого-либо класса. В этот момент создается объект и вызывается конструктор. В нашем примере, конструктору передается 0 в качестве стартовой позиции трансформера (это и есть вышеупомянутая инициализация). Ключевое слово new в некоторых языках отсутствует, и конструктор вызывается автоматически при попытке вызвать класс как функцию, например так: Transformer().
7. **Как вы определяете и используете статические методы в ООП? Статические методы** — это методы, которые принадлежат самому классу, а не экземплярам этого класса. Их можно вызвать без создания объекта.

**STATIC переменная**

-общая для всех объектов. Если мы у какого-то объекта ее изменим, то

она сразу изменится у всех

- чтобы работать с такой переменной даже не нужен экземпляр класса, можем

обращаться через класс целиком или обьекта

- ТАК НЕЛЬЗЯ будем описывать в паблик секции, но это дает сильную уязвимость, поэтому лучше использовать инкапсуляцию

не нужно, чтобы можно было изменять напрямую

- c помощью нее можно сделать простейший генератор айдишников

- прежде чем использовать статическую переменную ее необходимо инициализировать вне класса

**STATIC методы класса**

- имеет такое же поведение, н будет один единственный для всех экземпляров

класса и мы можем обращаться к нему не через обьект, а через весь класс

- так как работать с методом мы хотим так же на уровне класса, то необходимо делать его статическим

- когда мы хотим обратиться к нестатическим переменным, то НЕЛЬЗЯ использовать слово this

его можно использовать только внутри нестатической функции члена, т.е. this обращается к конкретном уобъекту

- в статических методах работа с нестатическими полями запрещена (такие для которых необходимо создание

еонкретного объекта). Статический метод знает о том, к какому типу он принадлежит, но не привязан к конкретному объекту

- если хотим работаь с обьектом, то передаем либо ссылку либо указатель, в таком случае также можно

обращаться от класса, но передавать конкретный объект

1. **Что такое конструктор и почему он важен в ООП? Конструктор** важен в ООП, потому что он обеспечивает начальную инициализацию объекта. Иначе может заполнять поля мусором.
2. **Объясните разницу между методами/полями объекта и класса.**

**Объект** — это набор переменных и функций, как в традиционном функциональном программировании. Переменные и функции и есть его свойства.

**Атрибуты** — это переменные, конкретные характеристики объекта, такие как цвет поля или имя пользователя.

**Методы —** это функции, которые описаны внутри объекта или класса. Они относятся к определенному объекту и позволяют взаимодействовать с ними или другими частями кода**.**

1. **Как ООП поддерживает концепцию модульности и организации кода?** Объектно-ориентированное программирование (ООП) поддерживает модульность благодаря тому, что код разделяется на независимые модули (классы), каждый из которых инкапсулирует данные и поведение. Это позволяет организовать код, упрощая его поддержку, тестирование и повторное использование. Каждый класс может быть разработан, протестирован и обновлен независимо от других, что улучшает структуру и читаемость программы.
2. **Определить наследование и его роль в ООП.** Наследование — это механизм ООП, который позволяет одному классу (подклассу) унаследовать свойства и методы другого класса (суперкласса). Это помогает повторно использовать код, уменьшить дублирование и поддерживать иерархии классов, отражающие реальные отношения между объектами.
3. **Как подкласс наследует свойства и поведение суперкласса?** Подкласс автоматически наследует все открытые и защищенные поля (свойства) и методы суперкласса. Это значит, что он может использовать методы суперкласса, а также может добавлять новые или переопределять существующие методы для изменения поведения.
4. **Объясните термины «суперкласс», «подкласс», «базовый класс» и «производный класс».**

**Суперкласс (базовый класс)** — это класс, от которого наследуют другие классы. Он содержит общие свойства и методы.

**Подкласс (производный класс)** — это класс, который наследует свойства и поведение суперкласса и может добавлять собственное поведение или изменять существующее.

1. **Что такое переопределение метода и почему оно важно при наследовании?** Переопределение метода — это процесс, при котором подкласс изменяет реализацию метода, унаследованного от суперкласса. Это важно, поскольку позволяет подклассу предоставлять специфичную для него реализацию метода, поддерживая полиморфизм.
2. **Опишите концепцию отношения «is-a» при наследовании.** Отношение «is-a» означает, что объект подкласса является специфическим типом объекта суперкласса. Например, если «Кот» наследует «Животное», то «Кот» является «Животным», и это отношение описывает наследование.
3. **Как ООП предотвращает «проблему ромба» с помощью виртуального наследования?** Проблема ромба возникает при множественном наследовании, когда два подкласса наследуют один и тот же базовый класс, что приводит к дублированию свойств. Виртуальное наследование (в таких языках как C++) позволяет решить эту проблему, обеспечивая, что общий базовый класс будет создан только один раз, и все наследники будут использовать эту единственную копию.
4. **Каковы преимущества и потенциальные недостатки использования наследования?**

 Повторное использование кода.

 Упрощение поддержки.

 Создание иерархий классов. Недостатки:

 Чрезмерная зависимость от структуры суперклассов может усложнить изменение кода.

 Неправильное использование может привести к сложным и запутанным иерархиям классов.

1. **Приведите пример одиночного и множественного наследования.**

 **Одиночное наследование:** Класс «Кот» наследует класс «Животное».

 **Множественное наследование:** Класс «Летучая мышь» наследует классы «Млекопитающее» и «Летучее существо» (в языках, которые поддерживают множественное наследование, например, C++).

1. **Как ООП поддерживает создание иерархий с использованием наследования?** Иерархии создаются через наследование, где базовые классы определяют общие характеристики, а производные классы специализируют поведение, строя дерево или цепочку классов, которые логически отражают отношения между сущностями.
2. **Когда в ООП следует отдавать предпочтение композиции, а не наследованию?** Композиция предпочтительнее, когда отношения между классами не являются отношениями «is-a», а скорее «has-a» (например, машина «содержит» двигатель). Это делает код более гибким, так как композиция позволяет изменять компоненты без затрагивания всей системы.

АГРЕГАЦИЯ И КОМПОЗИЦИЯ

- они описывают отношения между классами

- а зависимости от того, как мы будем использовать вложенный класс, ил как ы вложим один в другой

это и будет либо агрегация либо композиуия

- и агрегация и композиция это фактически включение одного класса в другой

АГРЕГАЦИЯ позволяет ипользовать тот класс, который мы включаем в другой класс использовать еще

и в других местах с другими классами

- класс кепка жестко к человеку не привязан

- классы не зависят друг от друга

КОМПОЗИЦИЯ более строгий вариант агрегации. При композиции класс, который включается в другой класс без этого

этого класса не может существовать, используется только для его служебных целей (human and brain)

- жесткая привязка одного объекта к другому

1. **Дайте определение полиморфизму и его значению в ООП. Полиморфизм** — это возможность объектов разных классов отвечать на один и тот же вызов методов по-разному.Полиморфизм — это возможность объектов разных классов отвечать на один и тот же вызов метода по-разному. Это позволяет использовать один и тот же интерфейс для различных типов объектов, что делает систему гибкой и расширяемой.
2. **Объясните разницу между полиморфизмом времени компиляции и полиморфизмом времени выполнения.**

 **Полиморфизм времени компиляции (перегрузка методов)** позволяет использовать один метод с разными параметрами, и решение о вызове конкретного метода принимается на этапе компиляции.

 **Полиморфизм времени выполнения (переопределение методов)** подразумевает, что решение о вызове метода принимается во время выполнения программы, основываясь на типе объекта.

1. **Что такое перегрузка метода и как она обеспечивает полиморфизм?** Перегрузка метода — это способность создавать несколько методов с одним и тем же именем, но с разными параметрами. Это позволяет использовать один и тот же метод для разных типов данных, что является примером полиморфизма времени компиляции.
2. **Опишите понятие «интерфейс» в контексте ООП.**

Интерфейс — это набор методов, которые должен реализовать класс. Интерфейсы определяют, что класс должен делать, но не как он это делает. Это позволяет различным классам реализовать один и тот же набор методов по-разному.

1. **Как интерфейсы способствуют множественному наследованию и полиморфизму?** Интерфейсы позволяют классам реализовать несколько интерфейсов, таким образом поддерживая множественное наследование и полиморфизм, поскольку объекты могут использоваться через интерфейсы, независимо от их конкретной реализации.
2. **Приведите пример полиморфизма с использованием переопределения метода.**
3. **Что такое абстрактные классы и как они связаны с полиморфизмом?**

Абстрактный класс — это класс, который не может быть инстанцирован напрямую и содержит хотя бы один абстрактный метод. Полиморфизм реализуется через абстрактные классы, так как подклассы предоставляют конкретную реализацию абстрактных методов.

1. **Как полиморфизм повышает гибкость и расширяемость кода?** Полиморфизм позволяет работать с объектами различных классов через общий интерфейс. Это повышает гибкость, так как можно добавлять новые классы и функциональность, не изменяя существующий код.
2. **Что такое Множественное наследование** Множественное наследование — это механизм, при котором класс может наследовать свойства и методы от нескольких суперклассов одновременно.
3. **Как ООП позволяет вам писать более общий и многократно используемый код?** ООП позволяет создавать обобщенные классы, которые инкапсулируют общие для разных объектов свойства и поведение. Эти классы могут быть использованы многократно и расширены для специфичных сценариев через наследование и полиморфизм.
4. **Определить инкапсуляцию и ее роль в ООП.**

Инкапсуляция — это механизм сокрытия данных и предоставления интерфейсов для работы с ними. Она защищает данные от прямого доступа, позволяя изменять внутреннюю реализацию, не затрагивая внешний код.

1. **Как инкапсуляция помогает обеспечить сокрытие данных и контроль доступа?** Инкапсуляция позволяет скрывать данные класса, предоставляя доступ к ним через методы (getters и setters). Это обеспечивает контроль за тем, как и когда данные изменяются, что повышает безопасность и стабильность системы.
2. **Объясните термины «public», «private» и «protected» модификатор доступа в ООП. Public –** открыто всем

**Private –** защищен, разреш только самому классу или дружественным классам и функциям

**Protected –** через обьект недоступно, но при наследовании можно получить доступ

1. **Приведите пример инкапсуляции в классе.**
2. **Что такое методы getters и setters и почему они используются?** Так как в классах необходимым условием и главным принциоп м является инкапсуляция т.е. сокрытие данных, то необоходимо каким-то способам получать доступ к приватным полям класса. Так геттеры позволяют получить информацию о полях, а сеттеры установить в них новое значение.Т.е. мы скрываем данные от пользователя, даем лишь интерфейс, а геттеры и сеттеры используются для взаимодействия со скрытыми полями.
3. **Описать преимущества инкапсуляции с точки зрения удобства сопровождения и безопасности.** Инкапсуляция улучшает сопровождение кода, так как внутренние детали реализации скрыты от внешнего мира. Это позволяет менять внутреннюю реализацию класса без необходимости изменения кода, который его использует. Также инкапсуляция повышает безопасность, предотвращая несанкционированное или некорректное изменение данных, поскольку доступ к ним контролируется.
4. **Как инкапсуляция способствует снижению сложности кода?** Инкапсуляция снижает сложность, так как отделяет внутреннюю логику класса от его внешнего поведения. Внешние пользователи взаимодействуют с классом только через его методы, не зная о деталях его реализации. Это упрощает понимание и использование классов и позволяет сконцентрироваться на их интерфейсах.
5. **Какова связь между инкапсуляцией и сокрытием информации?** Сокрытие информации — это ключевая часть инкапсуляции. Оно заключается в том, что детали реализации объекта скрыты, а доступ к ним осуществляется только через определенные методы. Это позволяет защитить данные от внешнего вмешательства и обеспечивает стабильность кода.
6. **Можно ли добиться инкапсуляции без использования классов и объектов? Почему да или почему нет?** Технически, можно добиться частичной инкапсуляции без классов и объектов, используя функции и замыкания, как в функциональных языках программирования. Однако классы и объекты обеспечивают более естественный и явный способ инкапсуляции данных и поведения, позволяя контролировать доступ к ним через модификаторы доступа и методы.
7. **Как инкапсуляция улучшает сотрудничество между несколькими программистами, работающими над проектом?**

Инкапсуляция улучшает совместную работу, так как каждый программист может работать с классом, не беспокоясь о его внутренней реализации. Если изменения необходимы, они могут быть внесены внутри класса без необходимости изменения других частей кода, что минимизирует конфликты и ошибки.

1. раннее связывания, позднее связывание, интерфейс, композиция, агрегация, super, конструктор , отношение «is-a», отношение «has-a» проблема ромба, множественное наследование, getters и setters, шаблоны проектирования, cohesion, coupling, переопределение(override), перегрузка(overload), Static

**Раннее связывание (compile-time binding)** — это процесс, при котором связь между вызовом метода и его реализацией происходит во время компиляции программы. Это типично для перегрузки методов, когда компилятор заранее знает, какой метод вызывать в зависимости от типа параметров.  
**Позднее связывание (run-time binding)** — это процесс, при котором решение о вызове метода принимается во время выполнения программы, в зависимости от типа объекта. Это связано с полиморфизмом и переопределением методов в наследуемых классах, когда конкретная реализация метода выбирается в зависимости от того, объект какого класса был вызван.

МОДИФИКАТОРЫ ДОСТУПА ПРИ НАСЛЕДОВАНИИ

****

Ассоциация – это когда один класс включает в себя другой класс в качестве одного из полей. Ассоциация описывается словом «имеет». Автомобиль имеет двигатель. Вполне естественно, что он не будет являться наследником двигателя (хотя такая архитектура тоже возможна в некоторых ситуациях).  
  
Выделяют два частных случая ассоциации: композицию и агрегацию.