Инкапсуляция: Это как коробка, в которой хранятся ваши вещи. Вы скрываете важные детали и предоставляете только нужные методы для работы с ними.

Наследование: Это, когда один класс может наследовать свойства и методы другого класса, как родитель может передать свои особенности своему ребенку.

Полиморфизм: Это, когда разные объекты могут делать одну и ту же вещь по-разному. Например, разные животные могут издавать звуки, но каждый делает это по-своему.

Абстракция: Это упрощение сложных вещей. Вы скрываете лишние детали и концентрируетесь на самом важном.

Определеить главный класс-абстракцию и от него переопределять одну функцию по разому-полиморфизм

Принцип единственной ответственности: Каждый класс должен делать только одну важную вещь. Не стоит перегружать один класс слишком многими задачами.

Принцип открытости/закрытости: Классы должны быть открытыми для расширения (вы можете добавлять новые функции), но закрытыми для изменения (вы не изменяете существующий код).

Принцип подстановки Барбары Лисков: Если у вас есть базовый класс и его подкласс, то вы должны быть в состоянии использовать подкласс везде, где используется базовый класс, и программа все равно будет работать правильно.

Эти принципы помогают создавать чистый, гибкий и легко поддерживаемый код в объектно-ориентированном программировании.

SOLID

Принцип единственной ответственности оказывается самым

сложным для понимания. Принцип гласит: каждый класс должен

быть ответственен лишь за что-то одно. Если класс отвечает за решение нескольких задач, его подсистемы, реализующие решение

этих задач, оказываются связанными друг с другом. Изменения в

одной такой подсистеме ведут к изменениям в другой.

Принцип открытости  закрытости означает, что готовый

код должен легко видоизменяться путем добавления нового кода,

а не путем видоизменения уже имеющегося. Другими словами,

программные сущности (классы, модули, функции) должны быть

открыты для расширения, но не для модификации.

Принцип подстановки Барбары Лисков состоит в возможности использования классов-наследников вместо родительских

классов, от которых они образованы, без нарушения работы программы. Если оказывается, что в коде проверяется тип класса, значит принцип подстановки нарушается.

Принцип разделения интерфейса (здесь под интерфейсом подразумевается набор методов) направлен на устранение недостатков, связанных с реализацией больших интерфейсов. Создавайте

узкоспециализированные интерфейсы, предназначенные для конкретного клиента. Клиенты не должны зависеть от интерфейсов,

которые они не используют.

Принцип инверсии зависимостей реализует принцип наследования и принцип абстракции в ООП и означает, что объектом зависимости должна быть абстракция, а не что-то конкретное:

 модули верхних уровней не должны зависеть от модулей

нижних уровней. Оба типа модулей должны зависеть от абстракций;

 абстракции не должны зависеть от деталей, наоборот, детали должны зависеть от абстракций.

А именно Вместо того, чтобы верхний уровень программы (например, главный модуль или класс) напрямую зависел от конкретных нижних уровней (например, классов с бизнес-логикой), верхний уровень должен зависеть от абстракций (интерфейсов или абстрактных классов).

Пример: Допустим, у вас есть класс, который отправляет электронные письма. Вместо того, чтобы этот класс напрямую зависел от конкретного почтового сервиса (например, Gmail), он должен зависеть от абстракции "Почтовый сервис". Таким образом, вы можете легко заменить Gmail на другой почтовый сервис, не изменяя код верхнего уровня.

Абстракции (интерфейсы или абстрактные классы) не должны зависеть от конкретных деталей и реализаций. Вместо этого, конкретные реализации должны соответствовать абстракциям.

Пример: Если у вас есть абстракция "Автомобиль" с методом "едь", то конкретные типы автомобилей (например, седан, внедорожник) должны реализовывать этот метод, но абстракция "Автомобиль" не должна заботиться о деталях каждой конкретной марки автомобиля.

Итак, согласно принципу инкапсуляции, поля класса являются закрытыми (private), мы оставляем открытыми (public) только методы, необходимые для взаимодействия объектов и обращения к полям объекта. При этом удобно спланировать классы так, чтобы зависимость между ними была наименьшей, как принято говорить в теории ООП, было наименьшее зацепление (low coupling) между классами – об этом же гласит первый принцип SOLID (п. 1.4.1). Тогда структура программы сильно упрощается. Кроме того, такие классы удобно использовать как строительные блоки для создания других программ. Напротив, члены класса должны активно взаимодействовать друг с другом, как говорят, иметь тесную функциональную связность (high cohesion). Для этого в класс следует включать все методы, описывающие поведение моделируемого объекта, и только такие методы, ничего лишнего. Одно из правил достижения сильной функциональной связности, введенное Карлом Либерхером (Karl J. Lieberherr), получило название закона Деметры. Закон гласит: «В методе m() класса А следует использовать только методы класса А, методы классов, к которым принадлежат параметры метода m(), и методы классов, экземпляры которых создаются внутри метода m() ».

Считается, что хорошо спроектированные модули должны обладать следующими свойствами: − функциональная целостность и завершенность — каждый модуль реализует одну функцию, но реализует хорошо и полностью; модуль самостоятельно (без помощи дополнительных средств) выполняет полный набор операций для реализации своей функции; − один вход и один выход — на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, т.е. реализуется стандартный принцип IPO — вход–процесс–выход; − логическая независимость — результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей; − слабые информационные связи с другими модулями — обмен информацией между модулями должен быть по возможности минимизирован.

наследование является неотъемлемой частью Java. При использовании наследования мы говорим: этот новый класс похож на старый класс. В коде Java это пишется как extends, после которого указывается имя базового класса. Тем самым мы получаем доступ ко всем полям и методам базового класса. Используя наследование, можно создать общий класс, который определяет характеристики, общие для набора связанных элементов (это абстрактный базовый класс). Затем мы можем наследоваться от него и создать новый класс, который будет иметь свои уникальные характеристики. Главный наследуемый класс в Java называют суперклассом. Наследующий класс называют подклассом. Получается, что подкласс − это специализированная версия суперкласса, которая наследует все члены суперкласса и добавляет собственные уникальные элементы.