ООП — подход к программированию, основанный на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объект — **Объект (экземпляр)** – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом. Сущность в комп. Пр-ве, обладающая определенным состоянием и поведением

Класс — универсальный тип данных, состоящий из набора полей и методов для оперирования своим содержимым. Шаблон описания сущности, определяющий состояние и поведение, а также интерфейс для взаимодействия с ним(контракт). Класс – элемент ооп описывающий интерфейс и его реализацию.

Основная задача ООП — сделать сложный код проще. Для этого программу разбивают на независимые блоки, которые мы называем объектами. Блоки можно повторно использовать. Упрощенный перенос объектов реального мира в программу.

В структурном программирование тяжело связывать структуру с необходимым исполняемым кодом. Код и данные находятся раздельно, поэтому чужой код может использовать не свои данные.

**Абстра́кция** в [объектно-ориентированном программировании](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование) — это использование только тех характеристик [объекта](https://ru.wikipedia.org/wiki/Объект_(программирование)), которые с достаточной точностью представляют его в данной системе. Основная идея состоит в том, чтобы представить объект минимальным набором полей и методов и при этом с достаточной точностью для решаемой задачи.

Позволяет рассматривать необходимые объекты данных и операции, без необходимости вникать в несущественные детали.

Связана с интерфейсом. В интерфейсе мы определяем, каким поведением будет обладать объект, по сути выделяя основные характеристики и способы использования объекта.

Абстрагирование — выделение наиболее значимых характеристик.

Инкапсуляция — свойство, позволяющее объединить данные и методы, работающие с ними, а также скрыть детали реализации. Заключается в разграничении доступа различных частей программы к внутренним компонентам друг друга. Данные объекта обрабатываются своими собственными методами. Инкапсуляция нужна чтобы методы не могли работать с чужими данными. Методы объединены с данными в классе, и вызываются на данных конкретного объекта.

В процедурном было так

Различные части программы не имеют доступа к в внутренним компонентам друг друга. Они обращаются друг к другу через интерфейс.

Плюс —возможность изменять реализацию, не меняя интерфейс. мы скрываем внутреннюю структуру, чтобы клиентский код не привязывался к ней. Вместо этого мы предлагаем ему использовать методы, которые меняться не будут. Так же возможность использовать классы не вникая во внутреннюю структуру. Инкапсуляция позволяет ослабить связи между объектами.

Наследование — свойство, позволяющее создавать классы на основе уже существующих классов частично или полностью заимствуя функциональность. Решает проблемы типизации, когда мы можеи точно определить к какому типу объект относится.

2 аспекта:

-наследование интерфейса (Интерфейс всегда наследуется)

-наследование реализации (Если она имеется. Например при наследовании интерфейсов нет реализации).

**Полиморфизм** —сбособность системы видоизменяться в зависимости от входных данных. Способность единообразного взаимодействия с различными типами данных.

Основная цель – повышения коэффициента повторного использования кода.

Истинный полиморфизм – выполняется один и тот же код для разных типов объектов.

Мнимый полиморфизм – для разных типов используется разный, но синтаксический похожий код. Не является полиморфизмом, и количество написанного кода не уменьшает.

**Полиморфизм подтипов** — свойство системы использовать объект, опираясь только на его интерфейс (контракт), не зная при этом фактического типа. (это возможность алгоритма работать со всеми подтипами одного общего типа, опираясь на свойства общего типа ). Интерфейс предка, реализация потомка. Такой подход позволяет подтипам реализовывать свое поведение и т.о. изменять поведение программы без перекомпиляции кода-клиента. (выз. Код исп. Obj не зная его пользвоат. типа)

**Параметрический полиморфизм** —позволяет определить функцию, метод или тип данных обобщенно, так что значения обрабатываются идентично вне зависимости от их типа.

подразумевает исполнение одного и того же кода для всех допустимых типов переменных (Дженерики). Использует аргументы на основе поведения, а не значения. Данный вид полиморфизма позволяет описывать вычисления в общем виде.

Другими словами, существует возможность описывать функции, которые будут работать с любыми типами данных. Такие функции называют параметрически полиморфными.

позволяет определять функцию, так что значения обрабатываются идентично для всех ДОПУСТИМЫХ типов.

возможность алгоритма работать с данными "не названных" типов, которым присвоены абстрактные имена. Т.е. типы являются Параметрами алгоритма, равно, как и его аргументы.

**AD-HOC** — (перегрузка, приведение типов) — данный вид полиморфизма позволяет определять функции и методы, которые будут иметь разную реализацию в зависимости от типов аргументов и их количества.(одинаковое имя, но различные сигнатуры). Перегруженными могут быть функции, методы, операторы.

данный вид полиморфизма позволяет описывать вычисления в общем виде. Другими словами, существует возможность описывать функции, которые будут работать с любыми типами данных. один и тот же метод будет работать с разными параметрами

В **объектно**-**ориентированном** **программировании** **ассоциация** означает отношение между классами объектов, которое позволяет одному экземпляру объекта вызвать другой, чтобы выполнить действие от его имени. Это структурное отношение, поскольку определяет связь между объектами одного рода и объектами другого рода и не моделирует поведение.

Ассоциация означает, что объекты двух классов могут ссылаться один на другой, иметь некоторую связь между друг другом.

**Агрегация** – это вид ассоциации, при котором объекты не сильно зависят друг от друга. (class A(class B)). отношение когда один объект является частью другого. Это просто ссылка на другой, самостоятельный объект. Он может продолжить функционировать, если ссылающийся объект прекратит существование. Время жизни объекта на который ссылаются не зависит от времени жизни ссылаемого.

**Композиция** – это вид ассоциации, при котором объекты сильно зависят друг от друга. **Композиция** подразумевает, что ассоциированный объект – «подобъект», составная часть, не имеющая смысла без ссылающегося.

(class A{ class B{}}). Еще более «жесткое отношение, когда объект не только является частью другого объекта, но и вообще не может принадлежат еще кому-то. При удалении главного объекта удаляется и подчиненный.

**Наследование -** определяет отношение IS A, то есть "является".

**Расширение** – дочерний класс добавляет новые функциональные возможности к родительскому классу, но не меняет наследуемое поведение. При расширении дочерний класс не переопределяет ни одного метода базового класса, а добавленные методы слабо связаны с существующими методами родителя.

Интерфейс — (контракт). Зная, что определенный класс реализует некий интерфейс, мы получаем гарантию, что данный класс реализовал все правила (поведение), диктуемые интерфейсом.

набор средств, используемых для взаимодействия двух систем.

Плюсы:

Упрощение реализации.

Выделение разных подходов к моделированию.

Изолирование методов от иерархии наследования. В результате классы, которые не связаны между собой иерархически, могут реализовать один и тот же интерфейс;

Уменьшение связности кода.

Абстрактный класс — базовый класс, который не предполагает создания экземпляров по причине их бессмысленности.

«заготовка» класса, средство для повторного использования кода. описывает некое общее состояние и поведение, которым будут обладать будущие классы. создается с целью создания общего интерфейса между разными реализациями **классов.**

Абстрактный класс без абстрактных методов нужен чтобы не было возможности создавать экземпляр класса. Данный класс используется только в качестве базового.

В java есть три типа наследования:

* Наследование состояния
* Наследование реализации
* Наследование типа

Java не поддерживает множественное наследование состояния (не позволяет расширять более одного класса), то есть наследование.

но поддерживает множественное наследование реализации (на основе методов по умолчанию интерфейсов).

Множественное наследование типов реализуется с помощью интерфейсов. Объект может иметь несколько типов: тип своего класса и типы всех интерфейсов, которые реализует сам класс.

**Разница между абстрактным классом и интерфейсом:**

* Интерфейс описывает только поведение, но не состояние. Интерфейс не сообщает ничего про объект, который его реализовывает. Абстрактный класс описывает как поведение, так и состояние. Он описывает некий абстрактный объект.
* Абстрактный класс используется только в определенной иерархии, он связывает между собой классы имеющие близкую связь. Один и тот же интерфейс могут реализовывать классы, не имеющее ничего общего.
* Классы могут реализовывать сколько угодно интерфейсов, но наследоваться можно только от одного класса.

**Позднее (динамическое) и раннее(статическое) связывание**

При статическом связывании вызов метода связывается с телом во время компиляции. Статическое связывание обеспечивает лучшую производительность. Выполняется для методов объявленных static, private, final – так как эти методы нельзя переопределить. В этом случае используется информация о типе, т. е. для обнаружения метода используется тип ссылочной переменной. Для перегруженных методов (методы с одинаковым именем, но разными сигнатурами) также используется статическое связывание.

Динамическое связывание происходит во время выполнения. Данный вид связывания выполняется для виртуальных методов (в Java методы по умолчанию являются виртуальными). При динамическом связывании для нахождения нужного метода используется конкретный объект.