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).

**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.

Абстрагирование — выделение наиболее значимых характеристик.

Инкапсуляция — свойство, позволяющее объединить данные и методы, работающие с ними, а также скрыть детали реализации. (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)

Наследование — свойство, позволяющее создавать классы на основе уже существующих классов частично или полностью заимствуя функциональность.

Полиморфизм —сбособность системы видоизменяться в зависимости от входных данных.

**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 )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** — 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.

**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.

**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.

«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**