OOP

Object-oriented programming is a model that organizes software design around objects which interact with each other.

Basic terms

Class

Data type acting like blueprint for individual objects, attributes, and methods. And rules for interacting with this entity

Objects

Instances of class created with specifically defined data. Object has a state (fields) and behavior (methods).

Methods

Functions describing behaviors of object.

Attributes

Defined in class template and represent state of object. Object fields.

Abstract Class | Interface

	Abstract Class	Interface
Describes	Attributes, methods	Methods
For classes	With close connections (inheritance)	That could have nothing in common
Multiple Inheritance	✖	✔
Key words	Implements interface Extends class	Extends interface
Attributes	✔	✖
Methods without realisation	✔ (`abstract` keyword)	✔
Methods with realisation	✔	✔ (`default` keyword)
Constructor	✔	✖
Access modifiers	any	`public` (default), `private` for methods with realisation

Inheritance (“is-a” relationship)

We can create a new class based on existing class. The new class can reuse the code and behavior of the ancestor class, and it can also add new features or modify the existing behavior.

Types:

• Single (one parent, one child)
• Multi-level (child is created from another child)
• Multiple (many parents, one child)
• Hierarchical (one parent, many children)
• Hybrid (child extend several parents, where one or more of them is a combination of different types of inheritance)

Encapsulation (What happens in Vegas...)

By encapsulating a class's variables, only the methods of the class can access them. It protects the data from external access or modification.

Polymorphism

Subclasses can define their own behaviors and yet share some of the same functionality of the parent class.

Compile time polymorphism -- Method overloading

Class can have more than one method with the same name, but with different parameters

Run time polymorphism - Method overriding

An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

Abstraction

We exposing only an object's relevant details to the outside world. And hiding the implementation details. Reduces the code's complexity and makes it easier to use.

Not only Inheritance

Composition “has-a” (strong connection)

Building has a room. Containing object owns it. Objects' lifecycles are tied (if we destroy the owner object, its members also will be destroyed with it)

Aggregation “has-a” (medium connection)

Car and its wheels. We can take off the wheels, and they'll still exist. Doesn't involve owning. Lifecycles of the objects aren't tied: every one of them can exist independently of each other.

Association objects “know” each other (weak connection)

Not only Inheritance (cont)

Mother and child. The difference with aggregation is only logical: whether one of the objects is part of other or not.

SOLID principles

Single Responsibility

Class should have a single, well-defined responsibility and should not be responsible for multiple things.

Open-Closed

Software enteties (classes, modules, functions) should be open for extension but closed for modification. You should be able to add new functionality to class without changing its existing code.

Liskov Substitution

Objects of a subclass should be able to be used in the same way as objects of parent class without issues.

Interface Segregation

Classes shouldn’t have to implement interfaces that they don't need.

Dependency Inversion

High-level modules (i.e., classes depending on other classes) should not depend on low-level modules. Both should depend on abstra-ctions. So, it's easier to change implementation of low-level module without affecting high-level module.