Introductions to OOP Principles

What is OOP?

Object-Oriented Programming (OOP) is a programming paradigm in computer science that relies on the concept of **classes** and **objects**. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages, including JavaScript, C++, Java, and Python.

What are Class and Object?

A **class** is an abstract blueprint that creates more specific, concrete objects. Classes often represent broad categories, like `Car` or `Dog` that share **attributes**. These classes define what attributes an instance of this type will have, like `color`, but not the value of those attributes for a specific object.

Classes can also contain functions called **methods** that are available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific object type.

An object is referred to as a data field that has unique attributes and behavior. Everything in OOP is grouped as self-sustainable objects. An object is an instance of a class. Each object can have unique values to the properties defined in the class.

A class is a logical entity whereas an object is an physical entity. Class doesn't allocate memory but object allocate memory.

Why do we need OOP? (Advantages of OOP)

- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Polymorphism allows for class-specific behavior
- Easier to debug, classes often contain all applicable information to them
- Securely protects sensitive information through encapsulation

The *Four Pillars* of OOP

- **Inheritance**
- **Encapsulation**
- **Abstraction**
- **Polymorphism**

Inheritance:

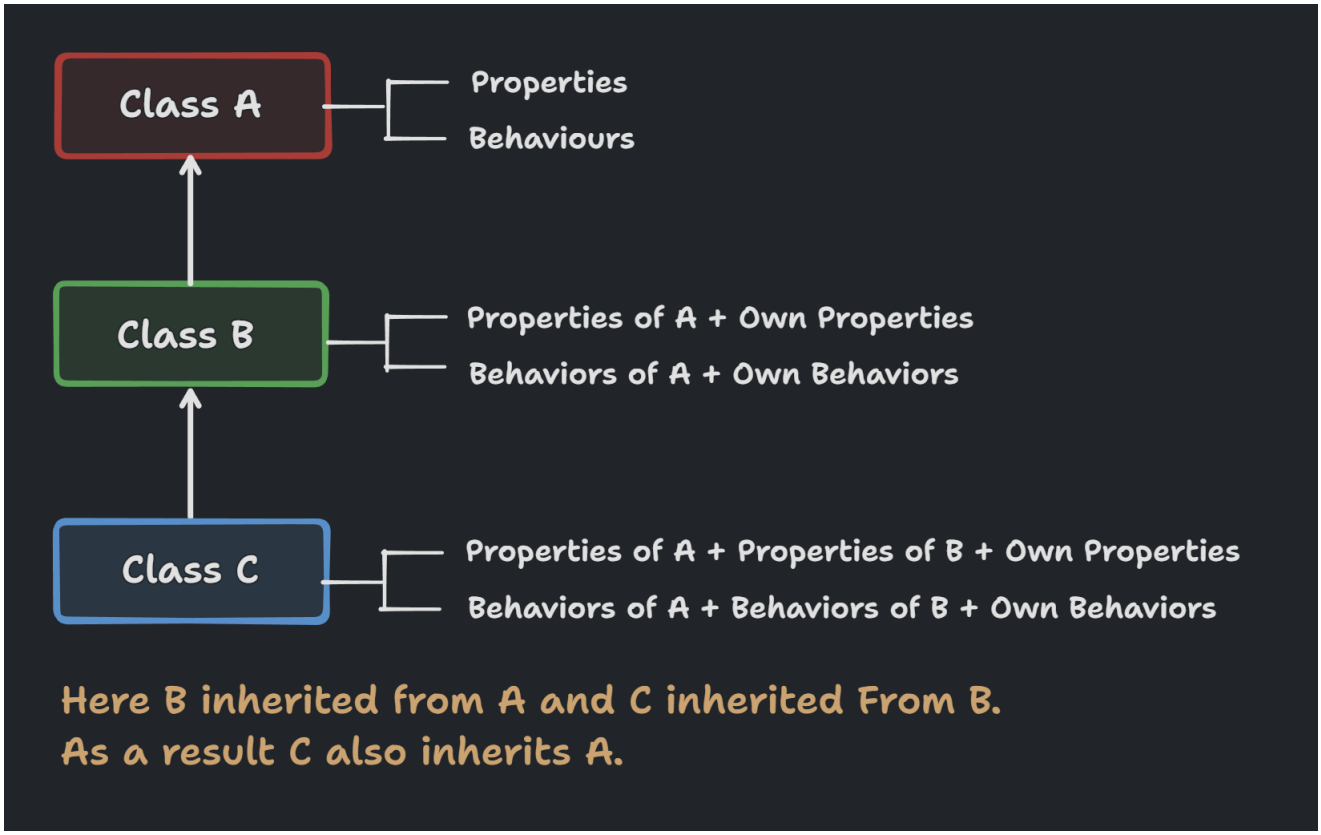
Inheritance allows classes to inherit features of other classes. Put another way, parent classes extend attributes and behaviors to child classes.

If basic attributes and behaviors are defined in a parent class, child classes can be created, extending the functionality of the parent class and adding additional attributes and behaviors.

For example, herding dogs have the unique ability to herd animals. In other words, all herding dogs are dogs, but not all dogs are herding dogs. We represent this difference by creating a child class `HerdingDog` from the parent class `Dog`, and then adding the unique `herd()` behavior.

The benefits of inheritance are programs can create a generic parent class and then create more specific child classes as needed. This simplifies programming because instead of recreating the structure of the `Dog` class multiple times, **child classes automatically gain access to functionalities within their parent class. Inheritance supports reusability**.

Parent classes are also known as superclasses or base classes. The child class can also be called a subclass, derived class, or extended class.



Encapsulation:

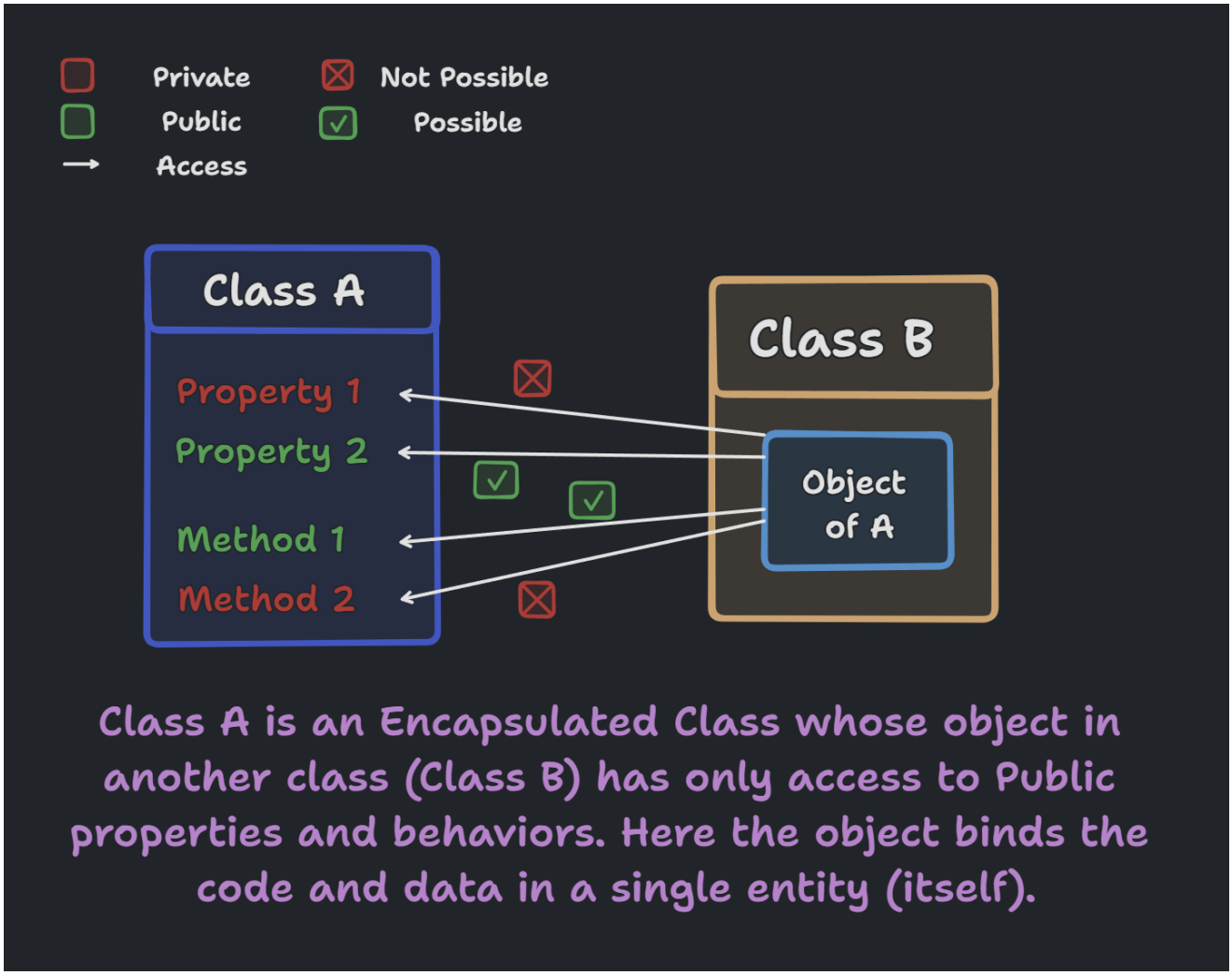
Encapsulation is a concept used in object-oriented programming to bundle data and methods into easy-to-use units. Encapsulation means containing all important

information **inside an object**, and only exposing selected information to the outside world. Attributes and behaviors are defined by code inside the class template.

Then, when an object is instantiated from the class, the data and methods are encapsulated in that object. Encapsulation hides the internal software code implementation inside a class and hides the internal data of inside objects.

To better understand encapsulation, view it as a medicine capsule that can't viewed from the outside. Similarly, in the realm of programming, encapsulation involves bundling data variables and the methods that manipulate the data into a single private unit, like a capsule. It conceals the inner workings and exposes only what is necessary.

Encapsulation adds **security**. Attributes and methods can be set to private, so they can't be accessed outside the class. To get information about data in an object, public methods & properties are used to access or update data. This adds a layer of security where the developer chooses what data can be seen on an object by exposing that data through public methods in the class definition.

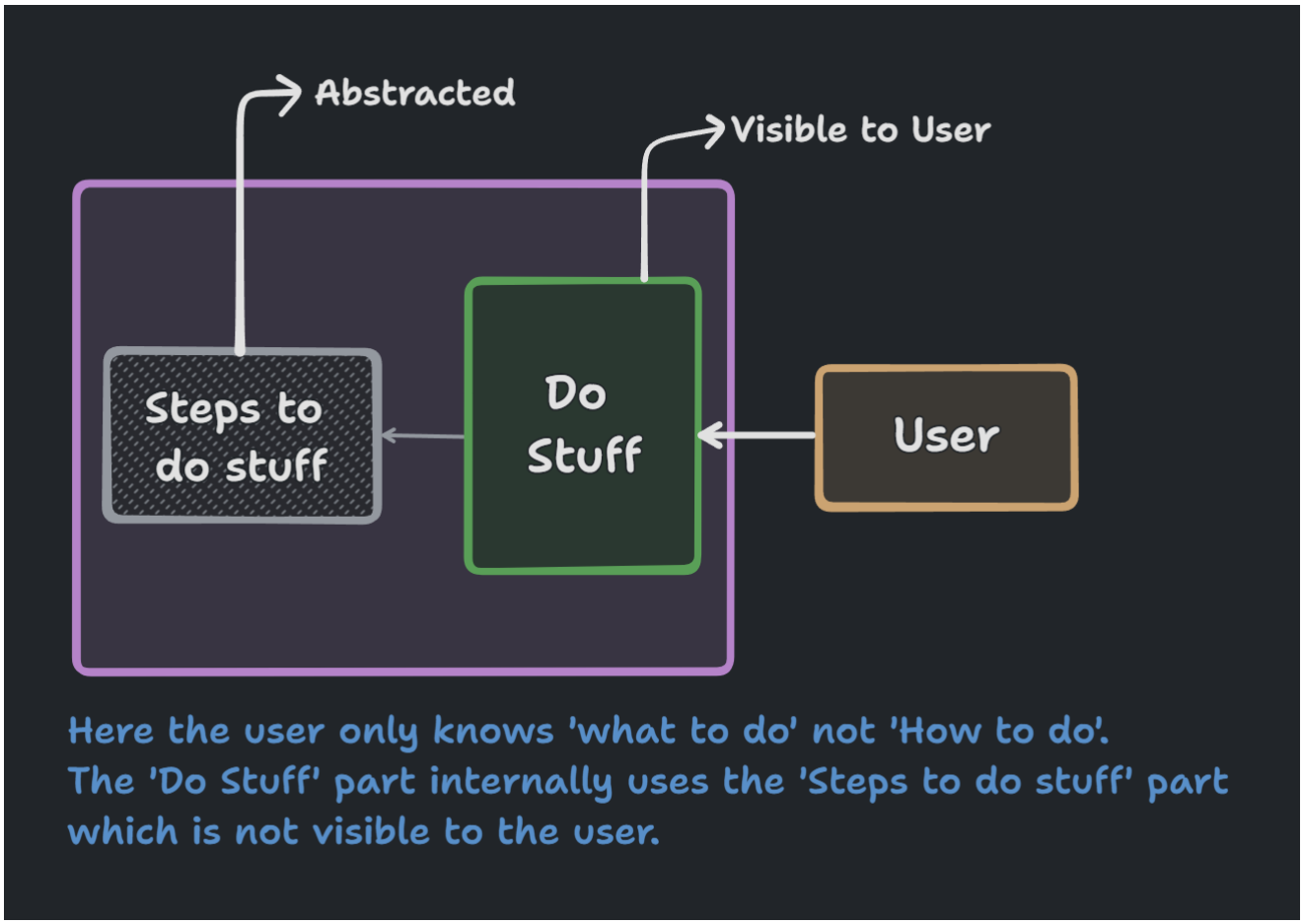


Abstraction:

Abstraction is an extension of encapsulation that uses classes and objects, which contain data and code, to hide the internal details of a program from its users. This is done by

creating a **layer of abstraction** between the user and the more complex source code, which helps protect sensitive information stored within the source code. The concept allows us to hide the implementation from the user but shows only essential information to the user.

Abstraction can also be explained using cars. Think of how a driver operates a vehicle using only the car's dashboard. A driver uses the car's steering wheel, accelerator, and brake pedals to control the vehicle. The driver does not have to worry about how the engine works or what parts are used for each movement. This is an abstraction – only the important aspects necessary for a driver to use the car are visible. Similarly, data abstraction allows developers to work with complex information without worrying about its inner workings. In this way, it helps to improve code quality and readability.



Polymorphism:

Polymorphism means designing objects to **share behaviors**. Using inheritance, objects can override shared parent behaviors with specific child behaviors. **The concept of polymorphism helps you to perform a single action in various ways. Polymorphism** aids the developer in reusing the program codes.

The term to **explain polymorphism** is such that the word "poly" indicates many and "morphs" indicates forms. Hence, it implies many forms.

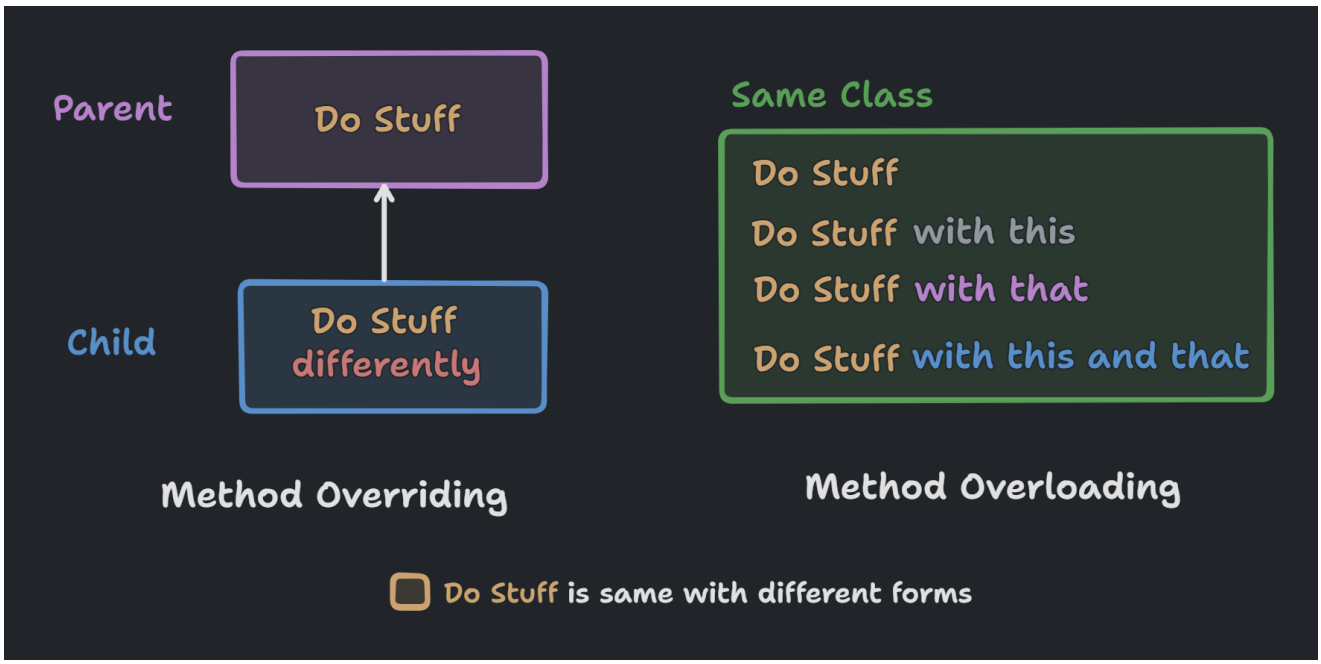
Polymorphism allows the same method to execute different behaviors in two ways: method overriding and method overloading.

Method Overriding:

Runtime polymorphism uses method overriding. In method overriding, a child class can implement method(s) differently than its parent class.

Method Overloading:

Compile Time polymorphism uses method overloading. Methods or functions may have the same name but a different number of parameters passed into the method call. Different results may occur depending on the number of parameters passed in. Method Overloading happens in same class.



Association, Aggregation and Composition:

Association:

Association is a relation between two separate classes which is established through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many. In Object-Oriented programming, an Object communicates to another object to use functionality and services provided by that object. ***Composition*** and ***Aggregation*** are the two forms of association.

Aggregation:

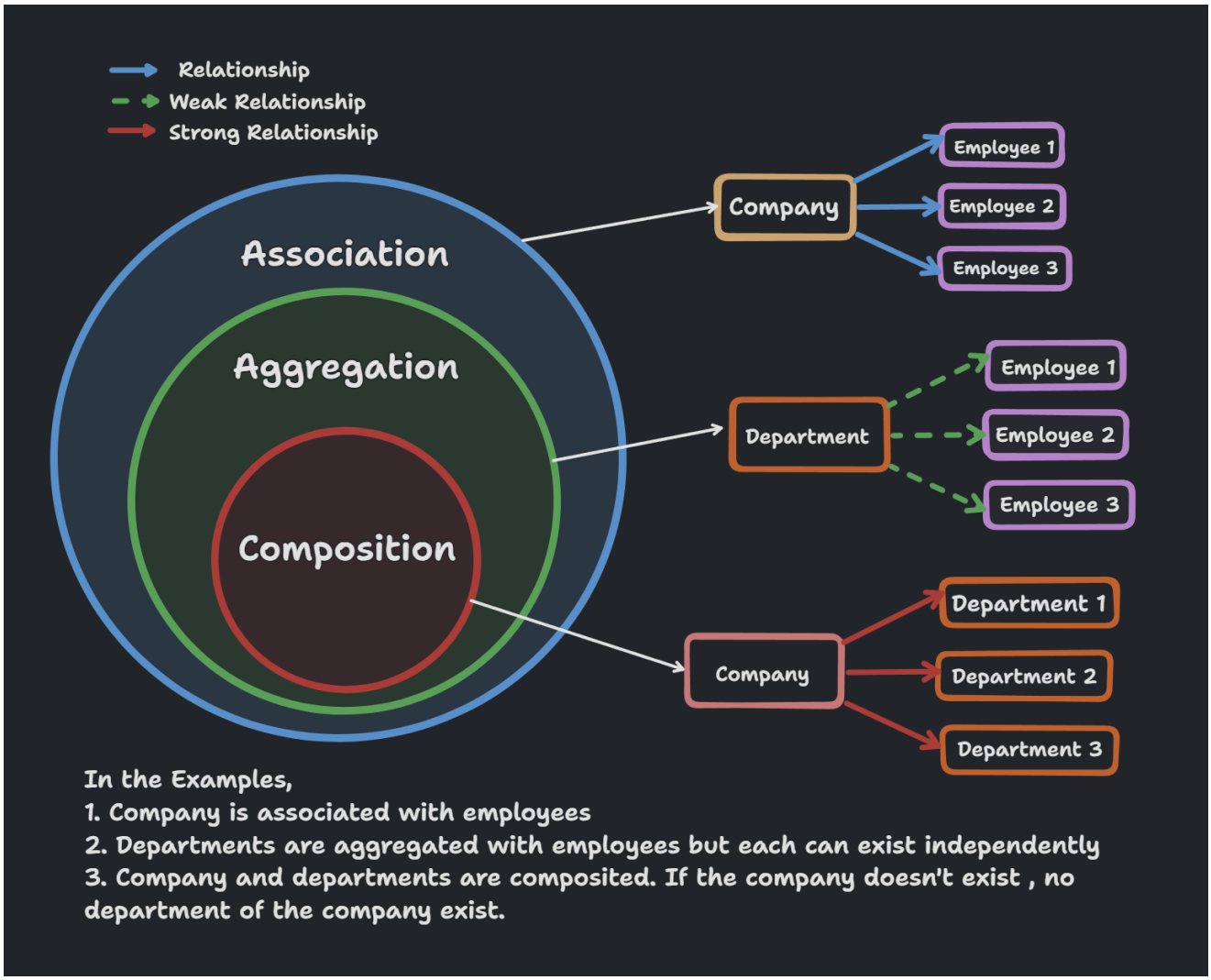
It is a special form of Association where:

- It represents Has-A relationship.
- It is a *unidirectional association* i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, *both entries can survive individually* which means ending one entity will not affect the other entity.

Composition:

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

- It represents *part-of* relationship.
- In composition, both entities are dependent on each other.
- When there is a composition between two entities, the composed object *cannot exist* without the other entity.



Inheritance represents IS-A relationship and Aggregation represents HAS-A relationship.

Sources:

Source 1
Source 2
Source 3

Source 4
Source 5