OOP

Object-Oriented Programming

OOP is a programming paradigm that relies on the concept of **classes** and **objects**.

Programming Language in OOP

- JavaScript
- C++
- Java
- Python

Class

An abstract blueprint used to create more specific, concrete objects.

Benefit

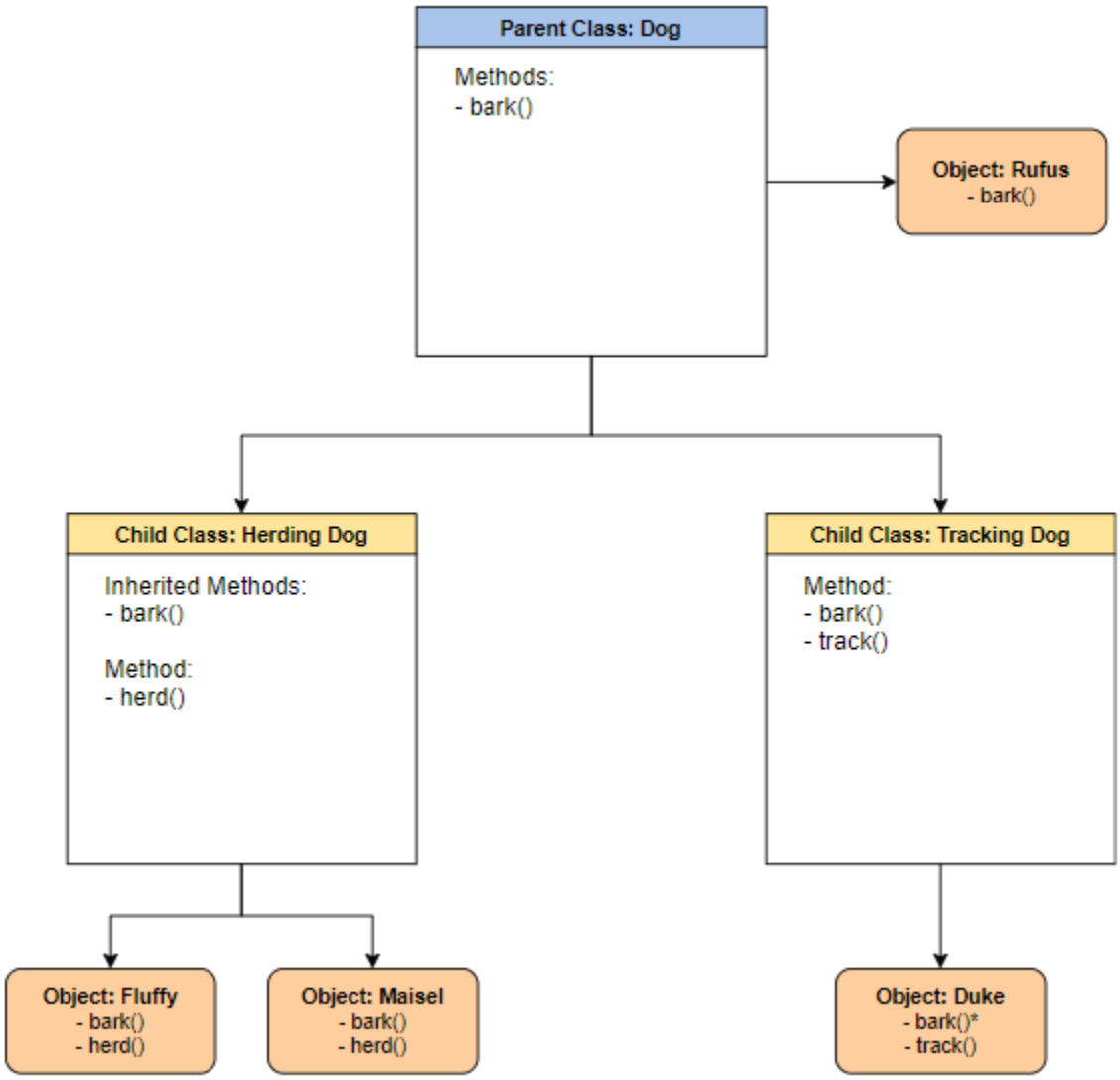
- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Allows for class-specific behavior through **polymorphism**
- Easier to debug, classes often contain all applicable information to them
- Secure, protects information through **encapsulation**

Structure

Grouping related information together to form a class structure makes the code shorter and easier to maintain.

- Create a parent class for all dogs

- Create child classes
- Add unique attributes and behaviors
- Create objects from the child class



The **Dog** Class

is a generic template, containing only the structure about data and behaviors common to all dogs.

Two child classes of **Dog**, **HerdingDog** and **TrackingDog**

have the inherited behaviors of **Dog (bark())** but also behavior unique to dogs of that subtype.

Objects of the HerdingDog type

to represent the individual dogs **Fluffy** and **Maisel**

Objects like **Rufus** that fit under the broad class of **Dog**

but do not fit under either **HerdingDog** or **TrackingDog**.

Building blocks of OOP

- Classes
- Objects
- Methods
- Attributes

Classes

- are essentially **user defined data types**
- contain **fields** for **attributes**, and **methods** for **behaviors**.

Objects

- are instances of classes created with **specific data**
- have **states and behaviors**. State is defined by **data**: things like names, birthday, and other information. Behaviors are **methods**, the object can undertake.

Attributes

- are the **information** that is stored.
- are defined in the **Class template**.
- When objects are **instantiated individual objects** contain data stored in the **Attributes field**.
- The state of an object is defined by the data in the **object's attributes fields**.

Methods

- represent behaviors, perform actions

- return information about an object, or update an object's data
- The code is defined in the class definition.
- often modify, update or delete data.
- Help promote reusability, and keep functionality encapsulated inside an object. (useful when debugging

Four Principles of OOP

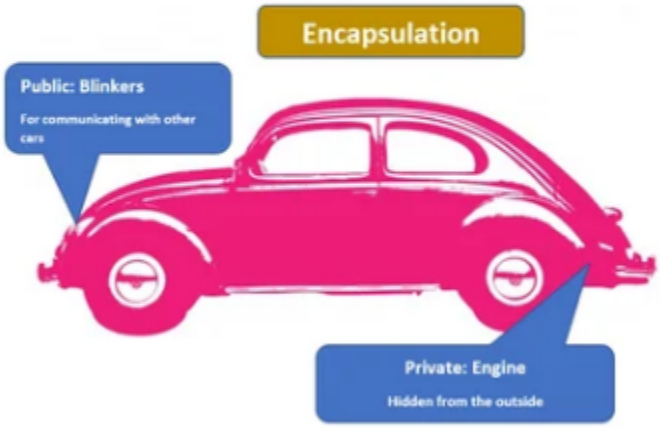
- **Encapsulation:** containing information in an object, exposing only selected information
- **Inheritance:** child classes inherit data and behaviors from parent class
- **Abstraction:** only exposing high level public methods for accessing an object
- **Polymorphism:** many methods can do the same task

Encapsulation

Encapsulation means containing all important information **inside an object**, and only exposing **selected information** to the outside world

- **Private/ Internal interface:** methods and properties, accessible from other methods of the same class.
- **Public / External Interface:** methods and properties, accessible also from outside the class.
- **Protected:** only accessible to **child classes**.

Example



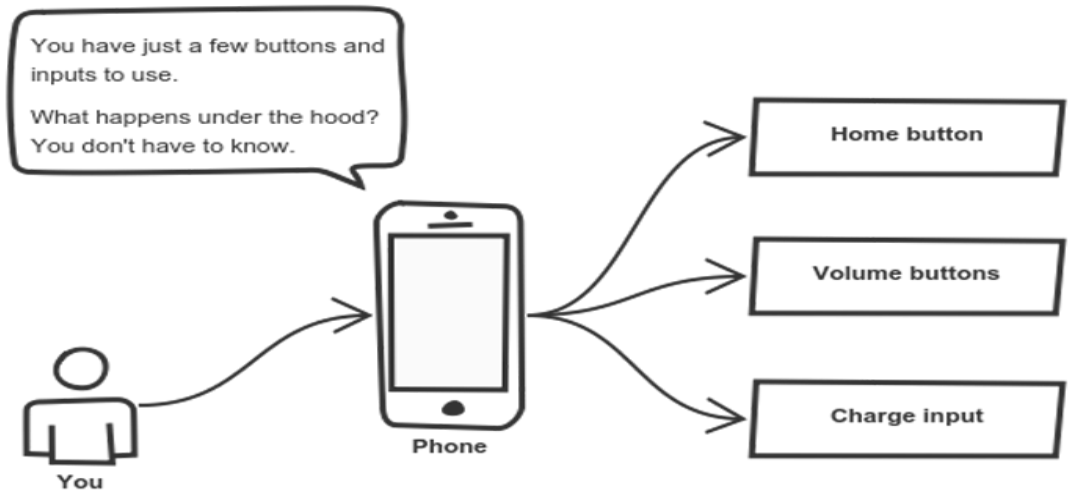
- Blinkers is exposed to indicate turns, are public interfaces (notice other drivers)
- The engine is hidden under the hood (avoid confusion)

Benefit

- **Adds security:** Only public methods and attributes are accessible from the outside
- **Protects against common mistakes:** Only public fields & methods accessible, so developers don't accidentally change something dangerous
- **Protects IP:** Code is hidden in a class, only public methods are accessible by the outside developers
- **Supportable:** Most code undergoes updates and improvements
- **Hides complexity:** No one can see what's behind the object's curtain!

Abstraction

- can be thought of as a natural **extension of encapsulation**.
- is using **simple classes** to **represent complexity**.



Cell phones are complex. But using them is simple

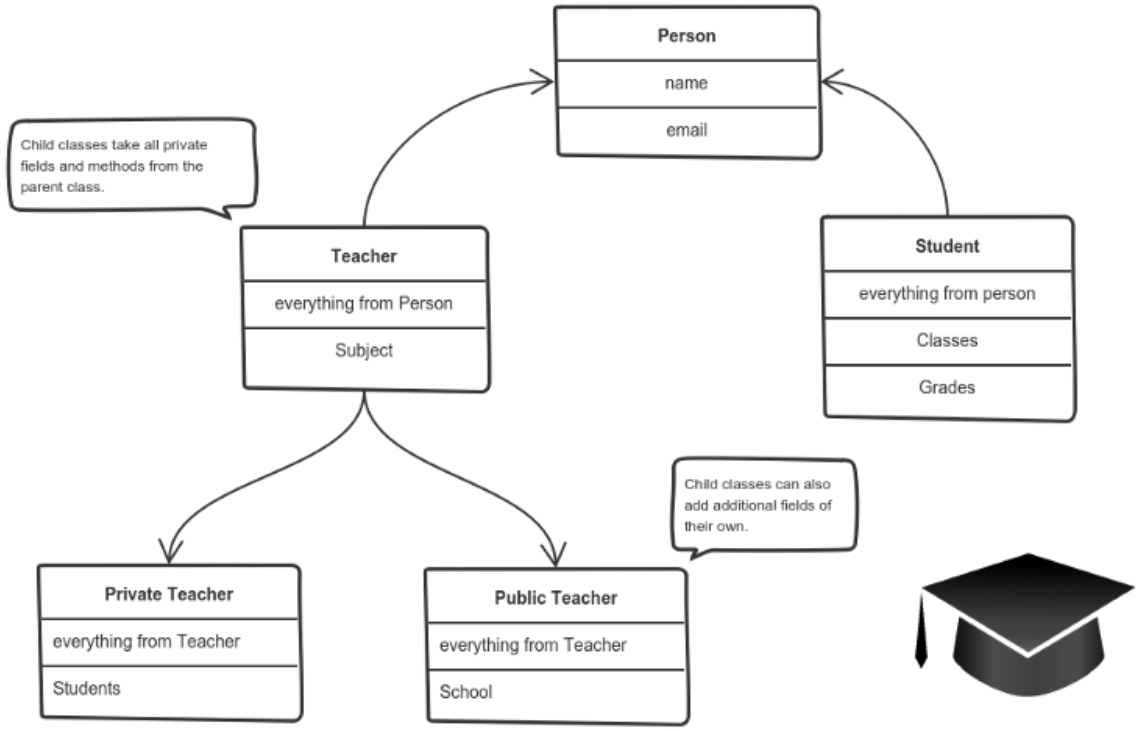
The users only interact with the phone by using only a few buttons, **implementation details** are **hidden**. Exposing that information to the driver would be **unnecessary**.

Benefit

- Simple, high level user interfaces
- Complex code is hidden
- Security
- Easier software maintenance
  - Code updates rarely change abstraction

Inheritance

- create a **(child) class** by deriving from another **(parent) class** => form a **hierarchy**.
- The child class reuses all fields and methods of the **parent class (common part)** and can implement its own **(unique part)**.



A private teacher is a type of teacher. And any teacher is a type of Person.

Example

- Program needs to manage public and private teachers, but also other types of people like students, this class **hierarchy** can be implemented.
- Each class adds only what is necessary for it while **reusing common logic** with the **parent classes**.

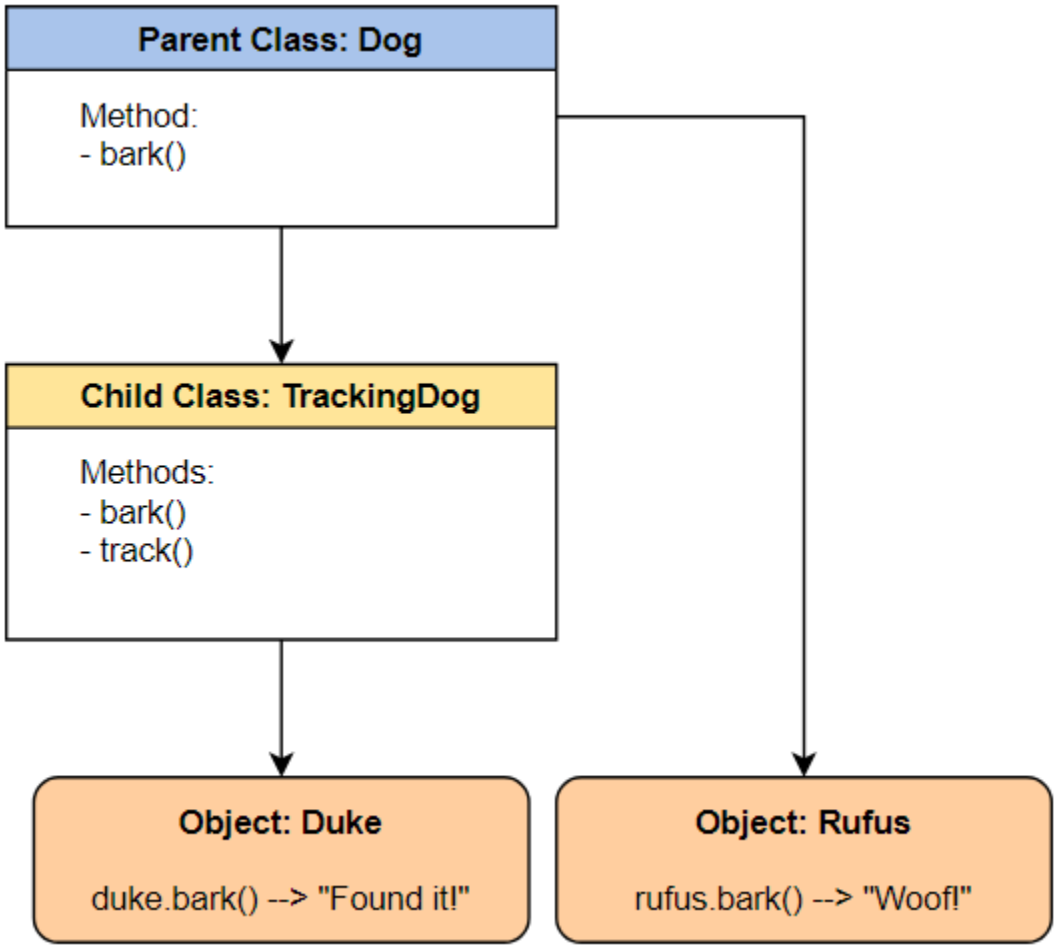
Polymorphism

- means designing objects to **share behaviors.**
- objects can **override** shared parent behaviors, with specific child behaviors
- allows the same method to execute **different behaviors** in two ways: **method overriding** and **method overloading**.

Method Overriding

Runtime polymorphism uses method overriding.

- A child class can provide a **different implementation** than its parent class.



TrackingDog's overriding the bark() method

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.

Benefit

- **Objects of different types** can be passed through **the same interface**
- Method overriding
- Method overloading