DAY 3

3) Inheritance

4 Polymorphism

} Pillars of OOPs

DAY 3 Inheritance and Polymorphism in OOPs

3) Pillar 3 : **Inheritance** → Definition:

'Inheritance is the capability of a class to derive properties and characteristics from another class'. In other words, it is a mechanism in which an objects acquired all the properties and behaviors of its parent object automatically.

https://medium.com/@derya.cortuk/inheritance-in-c-d60cd8b1805c [src ~ Medium article Derya Cortuk]

① What is Inheritance ?

→ Often, there is Parent  Child relationships among the objects.

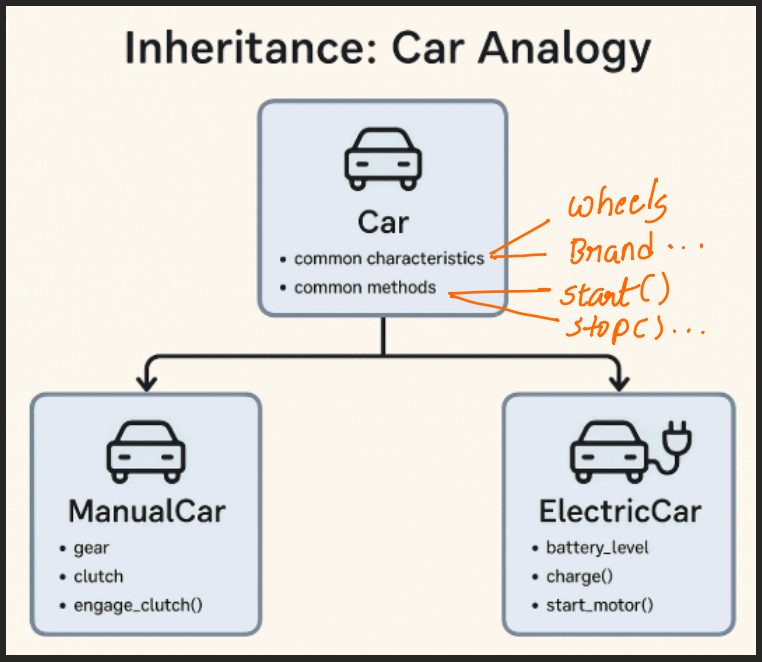
(parent) / (super class)

(child) / (sub class)

→ How do we represent this relationship in Objects ?

① Using Inheritance ✓✓

eg: The idea is that, Parent Object and Child Object share properties and behaviors. Take the analogy of a (Car).



If you observe, we are adding some extra features to a car to make it a Manual or Electric Car. In the process, you don't have to create all the methods and properties from scratch instead INHERIT the common methods from Parent class.

Code:

```
Class Car {
    // Common Members
    // Common Methods.
}
```

```
Class ManualCar : Public Car {
    ....
}
```

```
Class ElectricCar : Public Car {
    ...
}
```

Generalized Syntax:

```
class Name : Access Modifier Parent class {
    // Additional Members
    // Methods ...
}
```

Note:

① Public Inheritance : Maintain original access specifiers of parent class.

② Private, Protected alter accessibility.

④ Access Modifiers in Inheritance :-

The parent class might have members that could be public private or protected. when the inherited class (sub class) is:

① Public :

- Parent's Public members stay Public
- And Protected members stay Protected.

② Protected :

- Both Public and Private members become Protected.

③ Private :

- All inherited members become private.

Note: Private members of Parent class are Never Inherited.

The Question is How can you access the private members of the parent class?

- Basically, You have 2 choices.

Option 1 : Make use of setters and getters in Parent class

option 2 : Make the private Methods protected in parent class so that they are accessible by child class.

Summary:

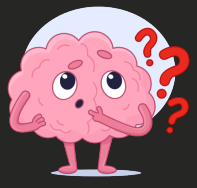
Access Modifier (Parent)	Outside Class (Some random)	Child Class (Inherited)
`private`	❌ Not accessible	❌ Not accessible directly
`protected`	❌ Not accessible	✅ Accessible
`public`	✅ Accessible	✅ Accessible

🔐 To **maintain encapsulation and flexibility**, use **private variables with public/protected getters and setters** in most cases.

```
/*
We know that real world Objects show inheritance relationship where we
have parent object and child object. child object have all the characters
or behaviors that parent have plus some additional characters/behaviors.
Like all cars in real world have a brand, model etc and can start, stop,
accelerate etc. But some specific cars like manual car have gear System
while other specific cars like Electric cars have battery system.

We represent this scenario of real world in programming by creating a parent class and
defining all the characters(variables) or behaviors(methods) that all cars
have in parent class. Then we create different child classes that inherits
from this parent class and define only those characters and behaviors
that are specific to them. Although objects of these child classes can
access or call parent class characters(variables) and behaviors(methods).
Hence providing code reusability.
*/
```

Disadvantage of Inheritance:



'Inheritance increases coupling (dependency) between base class and sub class. A change in Super class (Parent class) will affect all the child classes inherited from it) !!

4) Pillar 4 : **Polymorphism** → Definition:

"Polymorphism" simply means having many forms. In C++, polymorphism concept can be applied to functions & operators. A single function can work differently in diff scenarios. Similarly, an operator can work different when used in different content." [Src - Geeks for Geeks].

→ Derived from : 'Poly' (many) + 'Morph' (forms) = many forms.

→ One stimulus -> different responses based on object / situation.

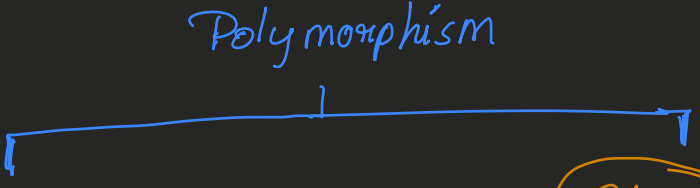
⭐ Two real-life scenarios :

Scenario 1 : → Different animals — Lion, Kangaroo, Human — } have a run() behavior

→ Each perform run differently.

Scenario 2 : Same human's run() differently based on context (tired Vs Energetic).

Types :-



Dynamic Polymorphism

Static Polymorphism

① Dynamic Polymorphism :

- Runtime Polymorphism / late binding. [Resolved at run time]
- Achieved via **Method overriding** / Function Overriding
- Achieved using "virtual functions" in C++.
- Same method signature is redefined in child class.

```
/*
Dynamic Polymorphism in real life says that 2 Objects coming from same
family will respond to same stimulus differently. Like in real world Manual
car and Electric car will respond to accelerate() differently.

To represent this in programming, we create a parent class that defines all
characters and behaviors that are generic to all child classes and are also same in
all child classes but make those methods abstract(virtual) that are generic to all
child classes but all child class will behave differently. Then those child class
will provide implementation details of these abstract methods the way they want.
*/
```

example :

```
class Car {
  virtual void accelerate() = 0; // Abstract
};

class ManualCar : public Car {
  void accelerate() override; // Manual-specific logic
};

class ElectricCar : public Car {
  void accelerate() override; // Electric-specific logic
};
```

→ Function overriding occurs when a derived class/child class defines one or more member functions of base class. That base function is said to be overridden. As shown above, the base function must be declared "virtual function".

② Static Polymorphism :

- Compile time Polymorphism / early binding (Resolved during Compile time).
- Achieved using Method OverLoading / Operator OverLoading
- Same Method name, different Parameter lists.
- Overloaded method — resolved at Compile time.
-

```
/*
Static Polymorphism (Compile-time polymorphism) in real life says that
the same action can behave differently depending on the input parameters.

For example, a Manual car can accelerate by a definition it want or by a
specific amount you request when provided as an argument. In programming,
we achieve this via method overloading: multiple methods with the same name
but different signatures.
*/
```

example :

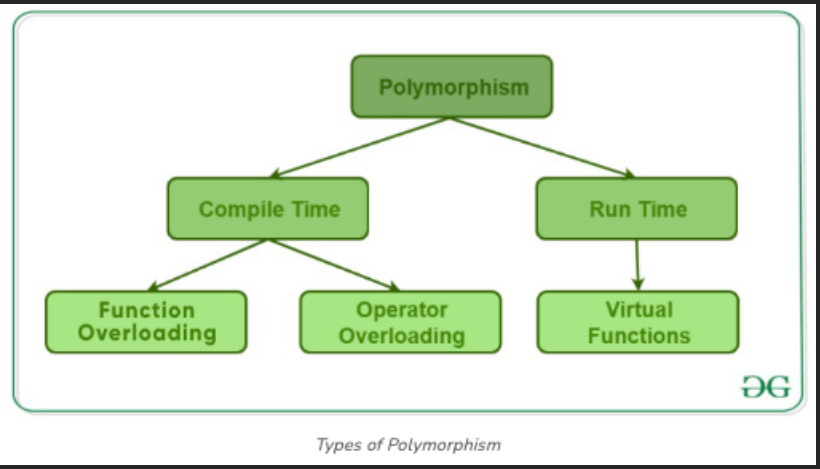
```
class ManualCar {
  void accelerate();              // no parameter
  void accelerate(int speed);     // with parameter
};
```

- Allow the behaviour to be choosen based on arguments passed.

Rules!!

- Method name : same
- Return type : Can be same / different (but not used for over loading.)
- How can parametres be different ?
  - Vary in number / type.

[Additional Resources]



Types of Polymorphism

(~GFG)

Why is Operator Overloading Not Supported by Java

Java | Object Oriented Programming | Programming

When a class has two or more methods by the same name but different parameters, at the time of calling based on the parameters passed respective method is called (or respective method body will be bonded with the calling line dynamically). This mechanism is known as ***method overloading***.

Operator overloading

Operator overloading is the ability to redefine the functionality of the operators. Programming languages like c++ supports operator overloading.

you can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Java does not support operator overloading due to the following reasons –

- **Makes code complex** – In case of operator overloading the compiler and interpreter (JVM) in Java need to put an extra effort to know the actual functionality of the operator used in a statement.
- **Programming error** – Custom definition for the operators creates confusion for the programmers especially the new developers. Moreover, while working with programming languages that support operator overloading the program error rate is high compared to others.
- **Easy to develop tools like IDEs** – Removal of operator overloading concept keeps the language simple for handling and process leading to a number of Integrated development environment in Java.
- **Method overloading** – The functionality of operator overloading can be achieved in Java using method overloading in Java in a simple, error free and clear manner.

[~ Tutorials Point]