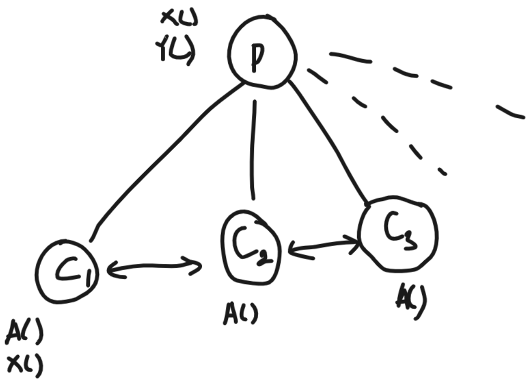
1. Strategy - Design Pattern

✏️ It focus on encapsulating a group of methods into separate classes of one interface.

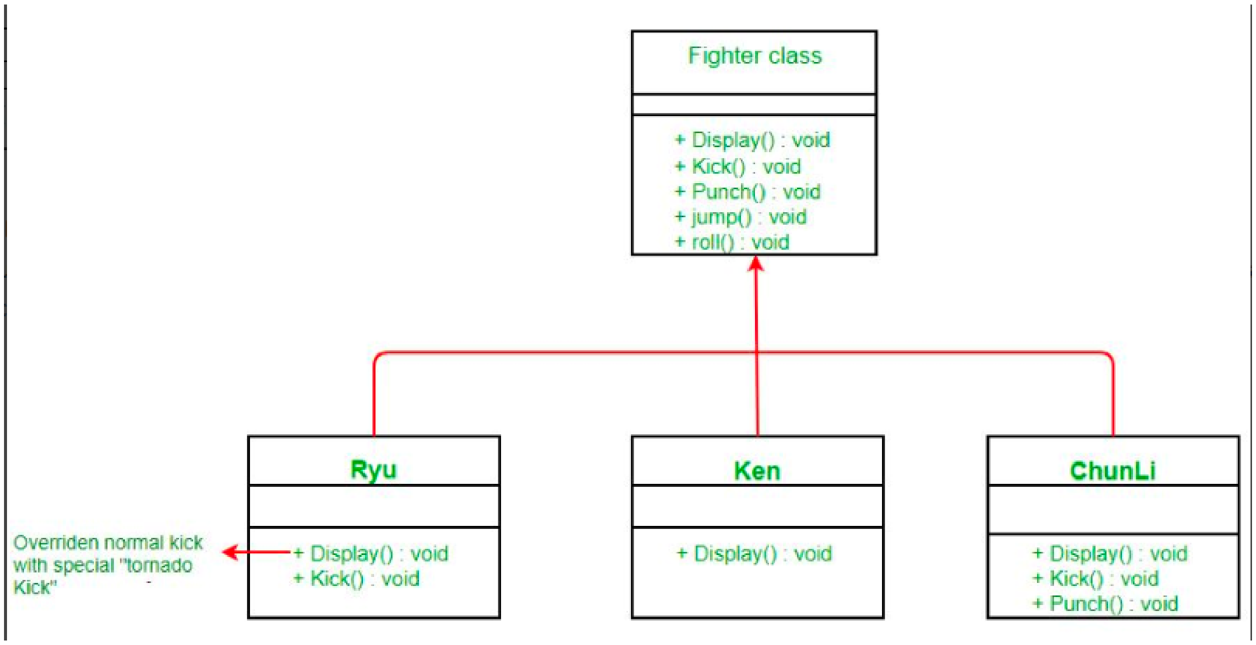
Problem: Consider a single-parent class with multiple child classes. If many child classes override some of the parent class methods, **but the children share the same overridden method among them, it can cause code reusability issues.** This is because the same logic is implemented repeatedly, making it difficult for the system to scale.



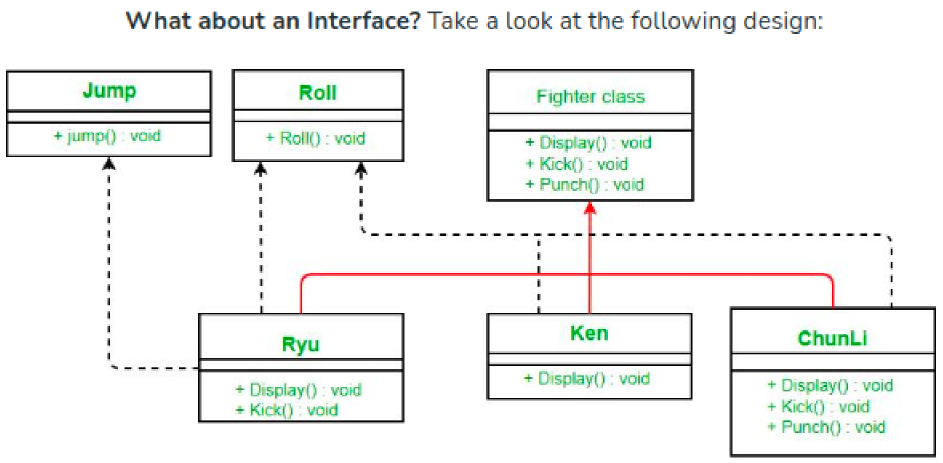
Like in this example, all the child classes share the same method A, which is not present in the parent class. Therefore, each one of them has to implement it again and again.

One more issue is that, even if some class don't use particular method of parent class, it will inherit that method.

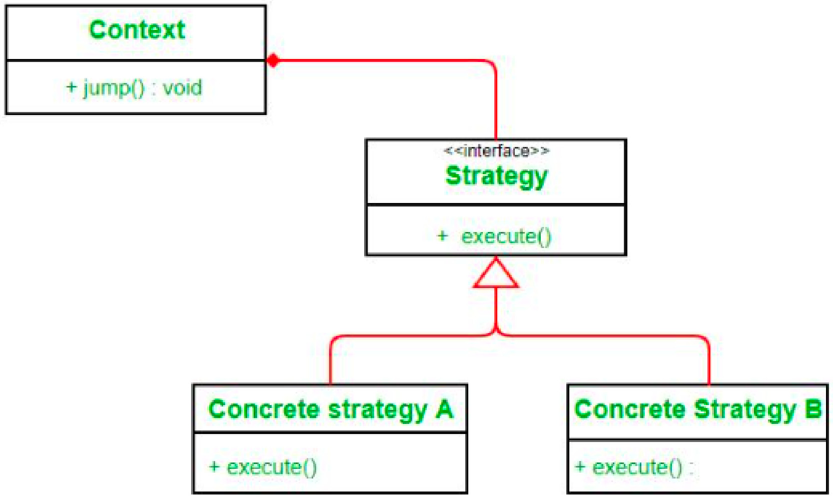
One solution to that can be to override the method, but then we have to do it for all the classes, which makes it less scalable.



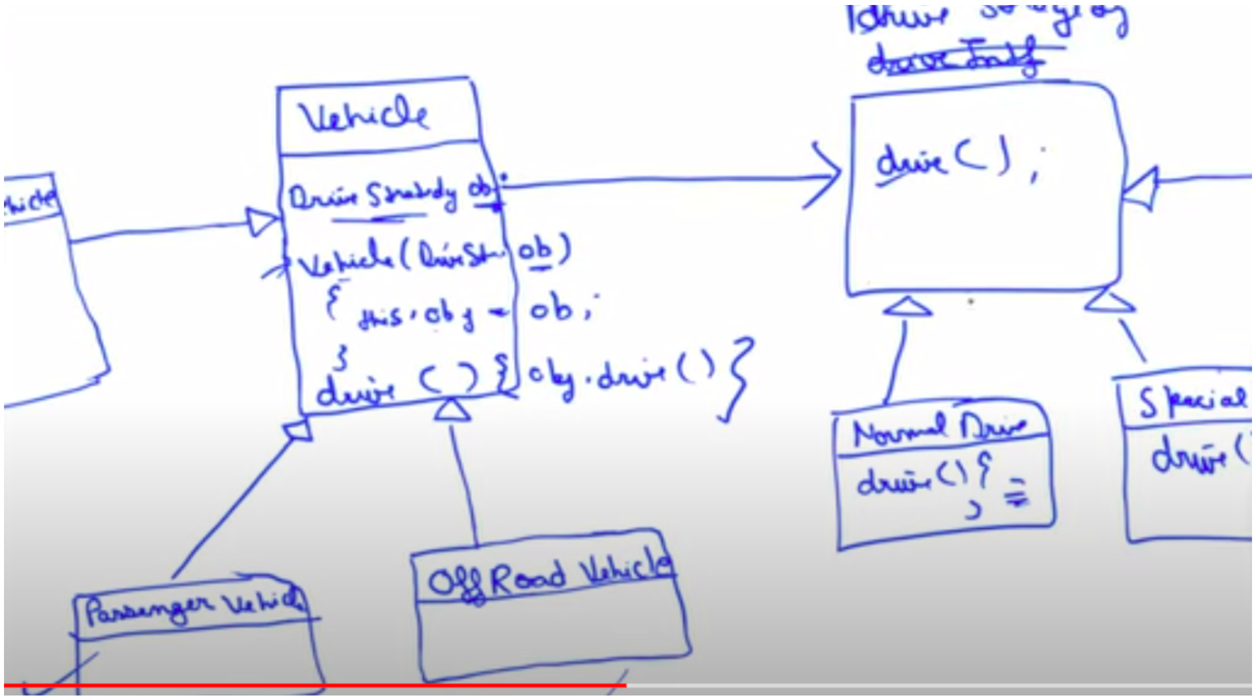
Other solution is to make an interface for these methods present in parent class. Again issues ⇒ code duplicity.



Now, the **best solution is to use a strategy design pattern.**



Example



Lets take a look at code

```cpp
#include <iostream>

// mode interface
class DriveMode
{
public:
    DriveMode() {}
    virtual ~DriveMode() {}

    virtual void drive() = 0;
};

// Drive mode 1
class NormalMode : public DriveMode
{
public:
    void drive()
    {
        std::cout << "Normal drive mode\n";
    }
};

// Drive mode 2
class BoostMode : public DriveMode
{
public:
    void drive()
    {
        std::cout << "Boost drive mode\n";
    }
};
// Similarly multiple modes can be defined

//  vehicle class,
class Vehicle
{
private:
    DriveMode *myModeObj; // So this statement make has-a relation with Mode Class
                            // we are not bounding our driveMode to any particular mode
                            // instead it will be decided by object of vehicle class when it will
                            // intialize.

public:
    Vehicle(DriveMode *mode)
    {
        this->myModeObj = mode;
    }

    void drive()
    {
        myModeObj->drive();
    }
};

// car
class Car : public Vehicle
{
public:
    Car() : Vehicle(new BoostMode())
    {
    }
};

// Bus
class Bus : public Vehicle
{
```

```
public:
    Bus() : Vehicle(new NormalMode())
    {
    }
};

// Bike
class Bike : public Vehicle
{

public:
    Bike() : Vehicle(new BoostMode())
    {
    }
};

int main(void)
{

    Car c1;
    c1.drive();

    Bus b1;
    b1.drive();

    Bike bi1;
    bi1.drive();

    return 0;
}
```

SO WHEN?

1. whenever we see that many sub classes are sharing same code which is not present in parent class then it is better to make parent contain a interface object of parent of that particular group of methods (usually similar in nature or same category methods) (for ex, here DriveMode)