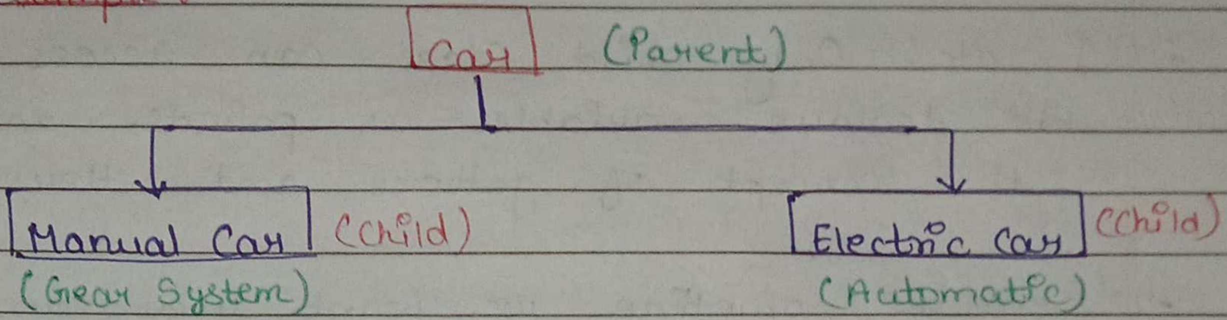
May
14/05/2025
Wednesday

DAY-3
LECTURE-3
Inheritance and Polymorphism

→ **Inheritance :** Parent - child Relationship. One object is related with another object

Example:



They both are car and exist in real life and relationship between them is parent-child and both exist in real world so we have represent in real life

Characteristics of Car
- → brand
- → model
- → Is Engine On
- → Current Speed

Start engine ()
Stop engine ()
accelerate ()
break ()

These all are exist in every car

Manual Car : Carry above features and some extra features.
→ Current Gear
→ Shift Gear

Electric Car : Specific features
- → battery Percentage
- → Charged Battery

Example:

```
Class Car {
    // Model
    // brand etc
}
```

Start engine();
Stop();

```
Class Manual Car : public Car
{
    // Shift Gear () { }
    Current Gear = 0;
}
```

Inherits all methods of Car and have its specific methods like shift Gear ();

```
Class Electric Car : public
                        Car
{
    // Charged Battery ();
    // Battery percentage ();
}
```

It can call ~~all~~ the methods of Car

In both Cases we don't need to redefine the methods like start engine() , Stop(). They both can access it from Car class and they both have their own specific methods.

Class Manual Car : Public Car
or
Private Car
or
Protected Car

Access modifier

Public Car ↓ Anonye can access

Private Car ↓ only class members and class methods can access

Protected Car ↓ like Private no one can access but its child ~~ed~~ Class can access ~~the~~ its properties

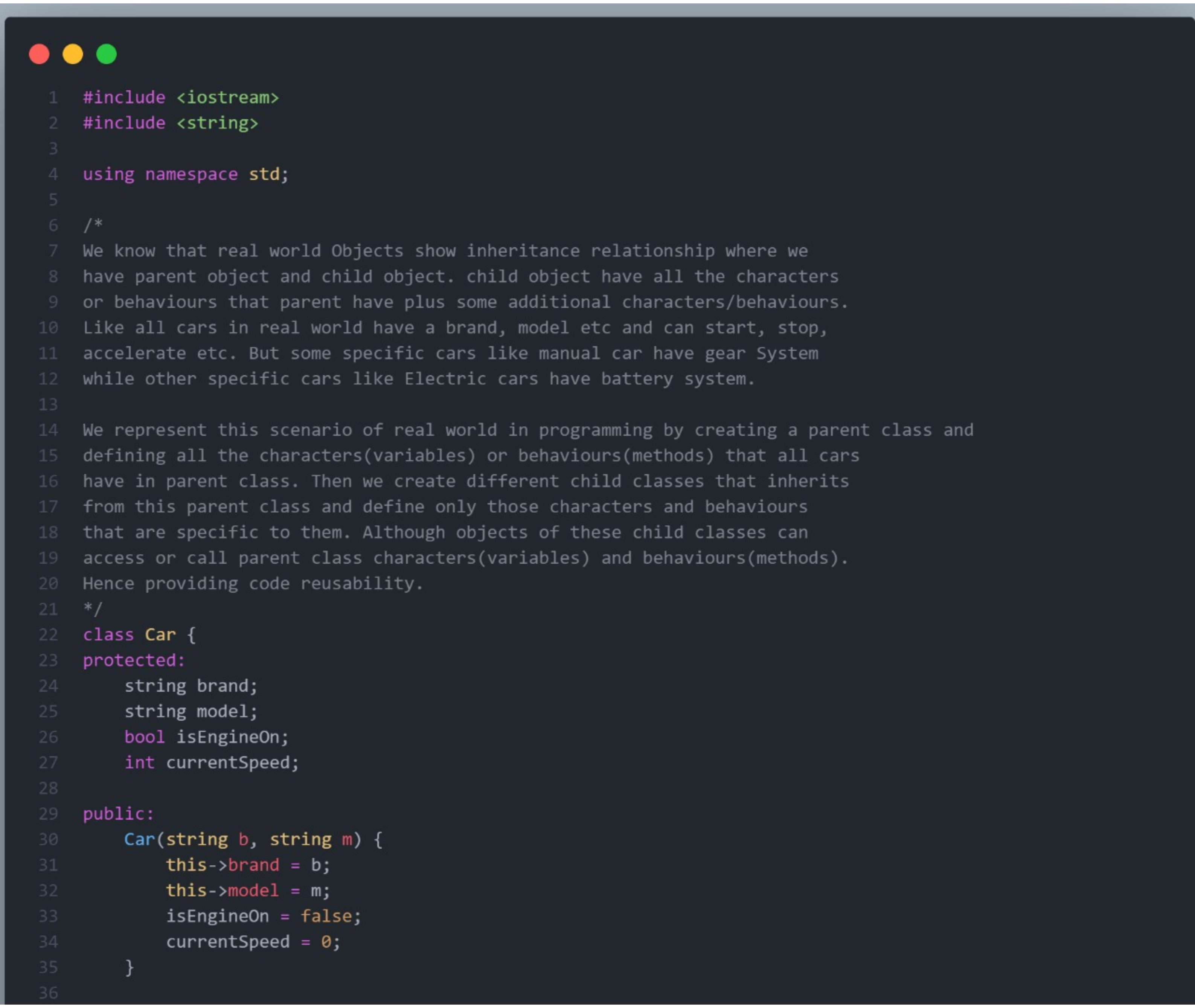
Syntax :

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

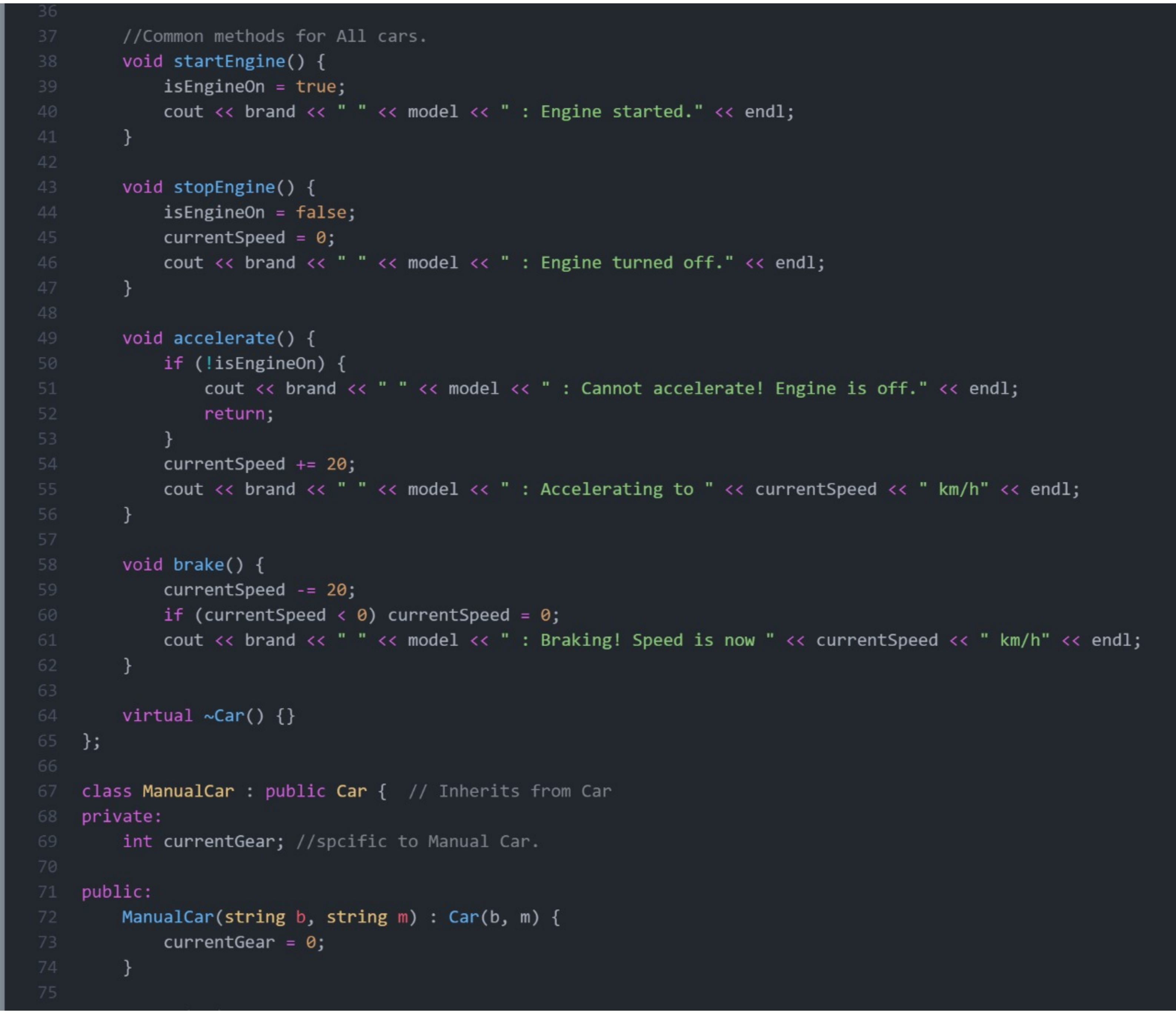
Accessibility Access Modifies in Inheritance

- Public
  - Public member stay public
  - Protected members stay protected
- Protected
  - Public and protected members become protected
- Private
  - All inherited members become private

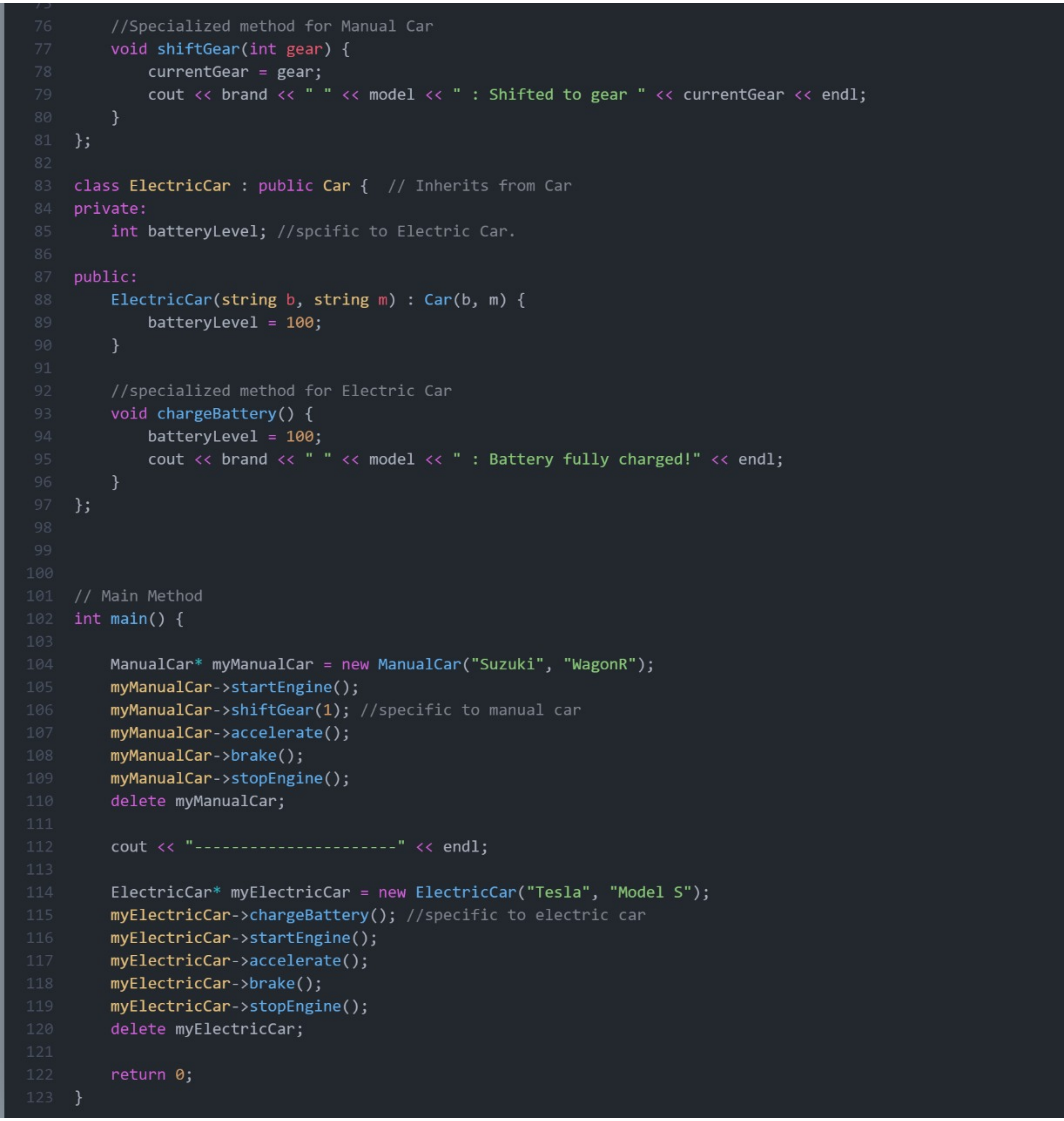
Private members of parent class are never inherited



```cpp
#include <iostream>
#include <string>

using namespace std;

/*
We know that real world Objects show inheritance relationship where we
have parent object and child object. child object have all the characters
or behaviours that parent have plus some additional characters/behaviours.
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 behaviours(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 behaviours
that are specific to them. Although objects of these child classes can
access or call parent class characters(variables) and behaviours(methods).
Hence providing code reusability.
*/
class Car {
protected:
    string brand;
    string model;
    bool isEngineOn;
    int currentSpeed;

public:
    Car(string b, string m) {
        this->brand = b;
        this->model = m;
        isEngineOn = false;
        currentSpeed = 0;
    }

```



```cpp

    //Common methods for All cars.
    void startEngine() {
        isEngineOn = true;
        cout << brand << " " << model << " : Engine started." << endl;
    }

    void stopEngine() {
        isEngineOn = false;
        currentSpeed = 0;
        cout << brand << " " << model << " : Engine turned off." << endl;
    }

    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += 20;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }

    void brake() {
        currentSpeed -= 20;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Braking! Speed is now " << currentSpeed << " km/h" << endl;
    }

    virtual ~Car() {}
};

class ManualCar : public Car {  // Inherits from Car
private:
    int currentGear; //spcific to Manual Car.

public:
    ManualCar(string b, string m) : Car(b, m) {
        currentGear = 0;
    }

```



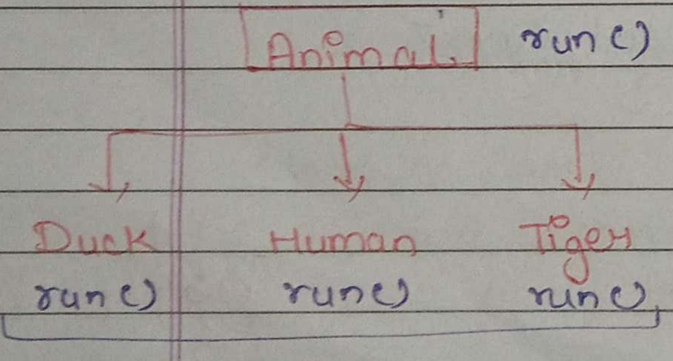
```cpp
    //Specialized method for Manual Car
    void shiftGear(int gear) {
        currentGear = gear;
        cout << brand << " " << model << " : Shifted to gear " << currentGear << endl;
    }
};

class ElectricCar : public Car {  // Inherits from Car
private:
    int batteryLevel; //spcific to Electric Car.

public:
    ElectricCar(string b, string m) : Car(b, m) {
        batteryLevel = 100;
    }

    //specialized method for Electric Car
    void chargeBattery() {
        batteryLevel = 100;
        cout << brand << " " << model << " : Battery fully charged!" << endl;
    }
};



// Main Method
int main() {

    ManualCar* myManualCar = new ManualCar("Suzuki", "WagonR");
    myManualCar->startEngine();
    myManualCar->shiftGear(1); //specific to manual car
    myManualCar->accelerate();
    myManualCar->brake();
    myManualCar->stopEngine();
    delete myManualCar;

    cout << "----------------------" << endl;

    ElectricCar* myElectricCar = new ElectricCar("Tesla", "Model S");
    myElectricCar->chargeBattery(); //specific to electric car
    myElectricCar->startEngine();
    myElectricCar->accelerate();
    myElectricCar->brake();
    myElectricCar->stopEngine();
    delete myElectricCar;

    return 0;
}
```

→ Polymorphism :

Poly → Many
Morphism → form

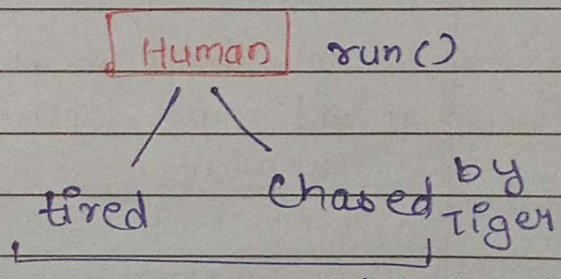
Some objects have many form

Scenario-1



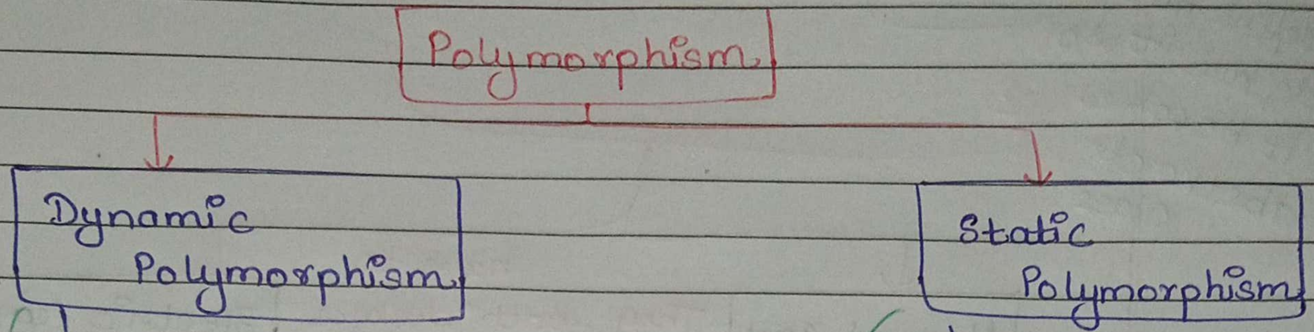
Each performs run() differently

Scenario-2



In both same human perform run differently

Polymorphism means same object perform differently based on situation



Dynamic Polymorphism → Method Overriding

→ Example Scenario-1

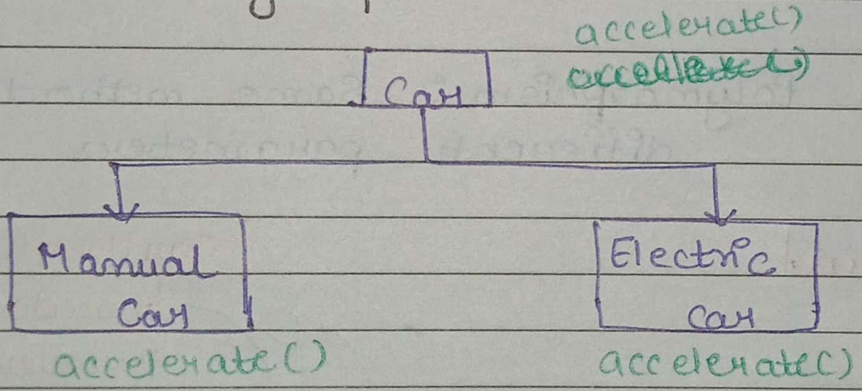
Animal can be duck, human, tiger and they ~~[illegible]~~ perform different reaction on ~~different~~ run.

Static Polymorphism → Method Overloading

→ Example Scenario-2

One object Human and one ~~[illegible]~~ method run(), if we change situation or parameter then it perform different

Dynamic Polymorphism:



→ Car has method accelerate but

→ Manual Car perform accelerate differently than the electric car.

→ Manual Car has Gear so it perform accelerate speedly or in electric car may be it perform accelerate slow.

→ Both use method accelerate but in their own way

We only define not declare it so it become abstract class

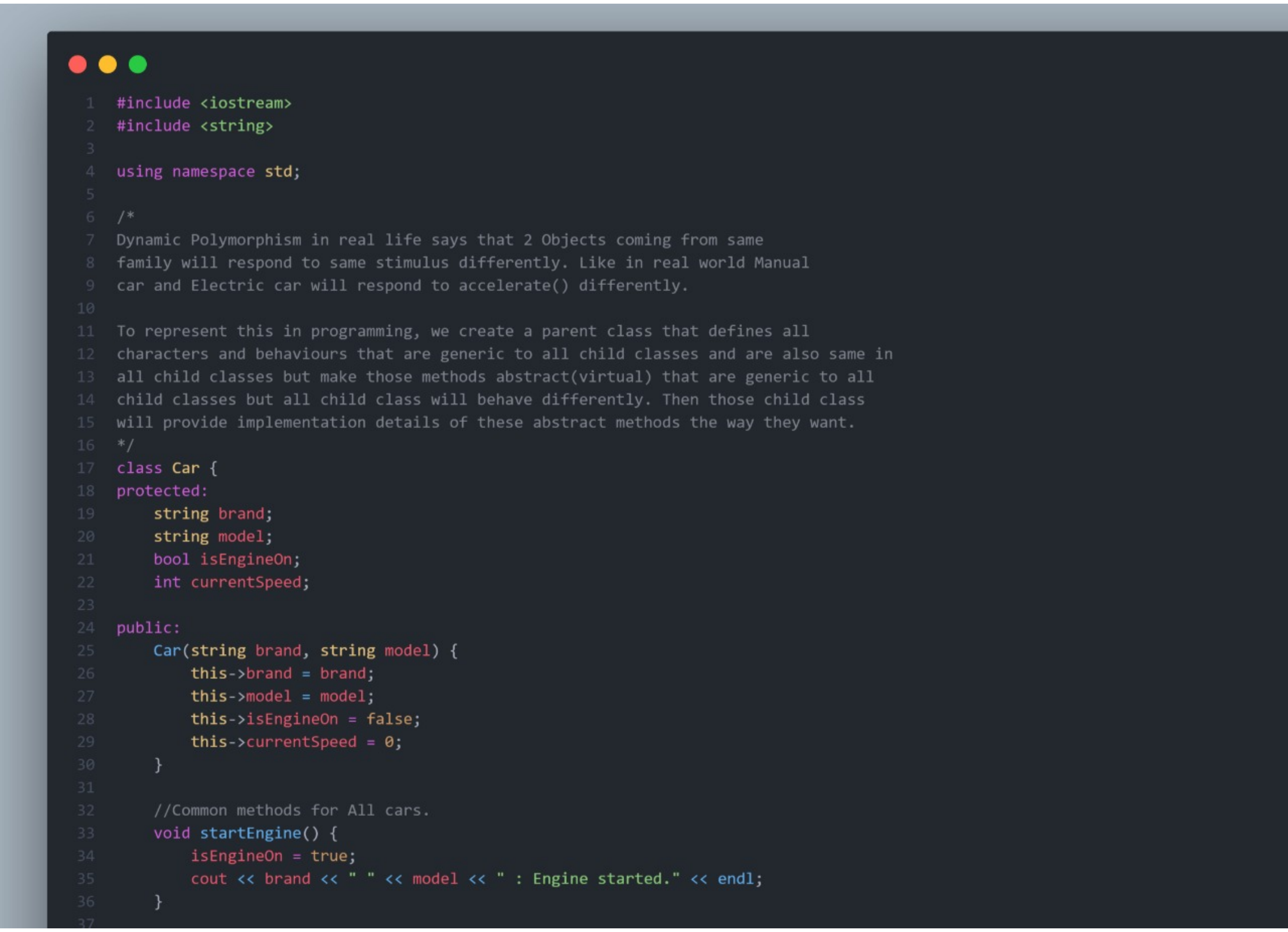
```
Class Car {

    virtual accelerate () = 0;   → Abstract Class

}
```

↓ Manual car   ↓ Electric Car

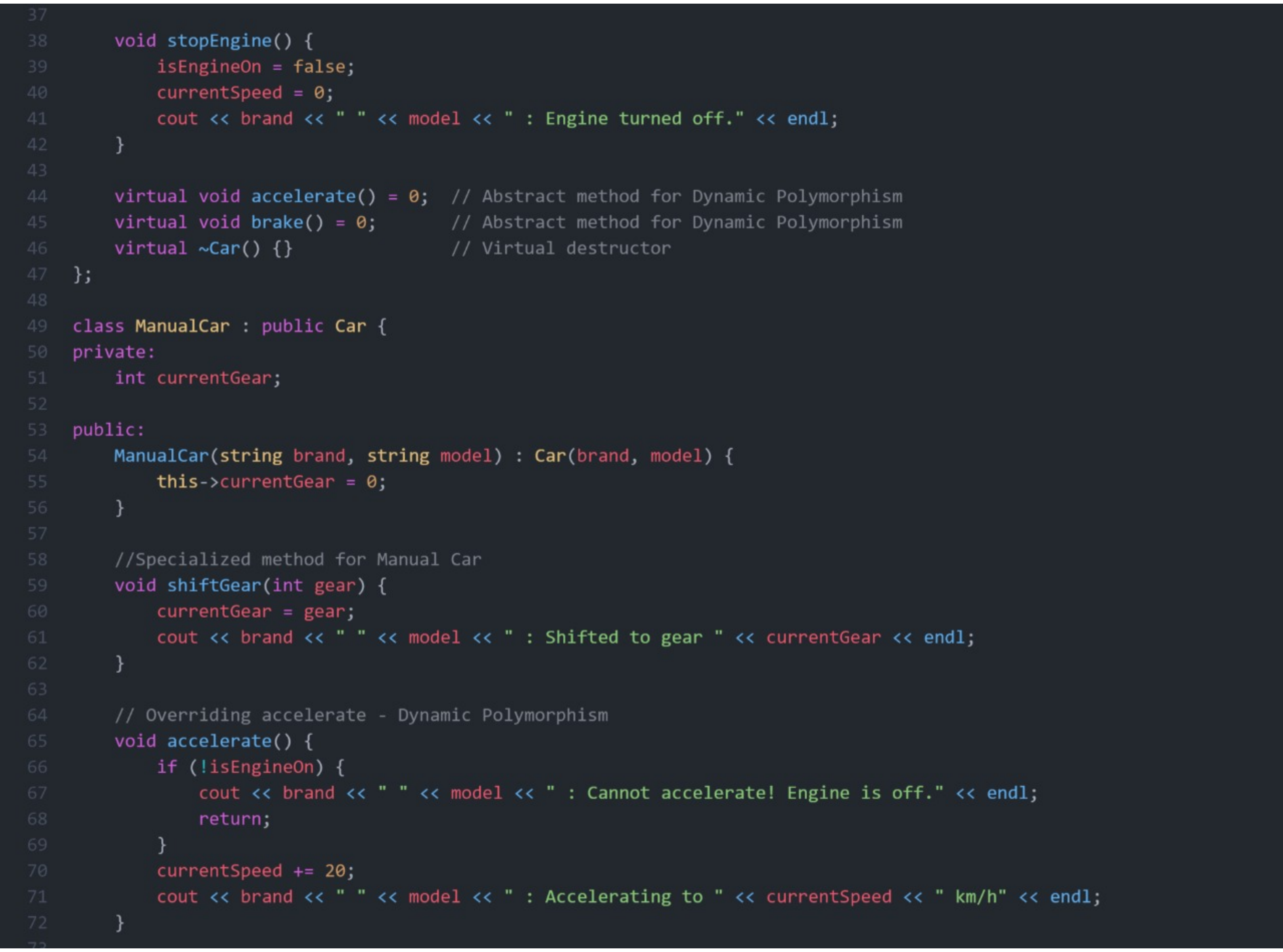
They both have to declare accelerate method

```
accelerate()                          accelerate()
{                                     {

    }                                     }
```

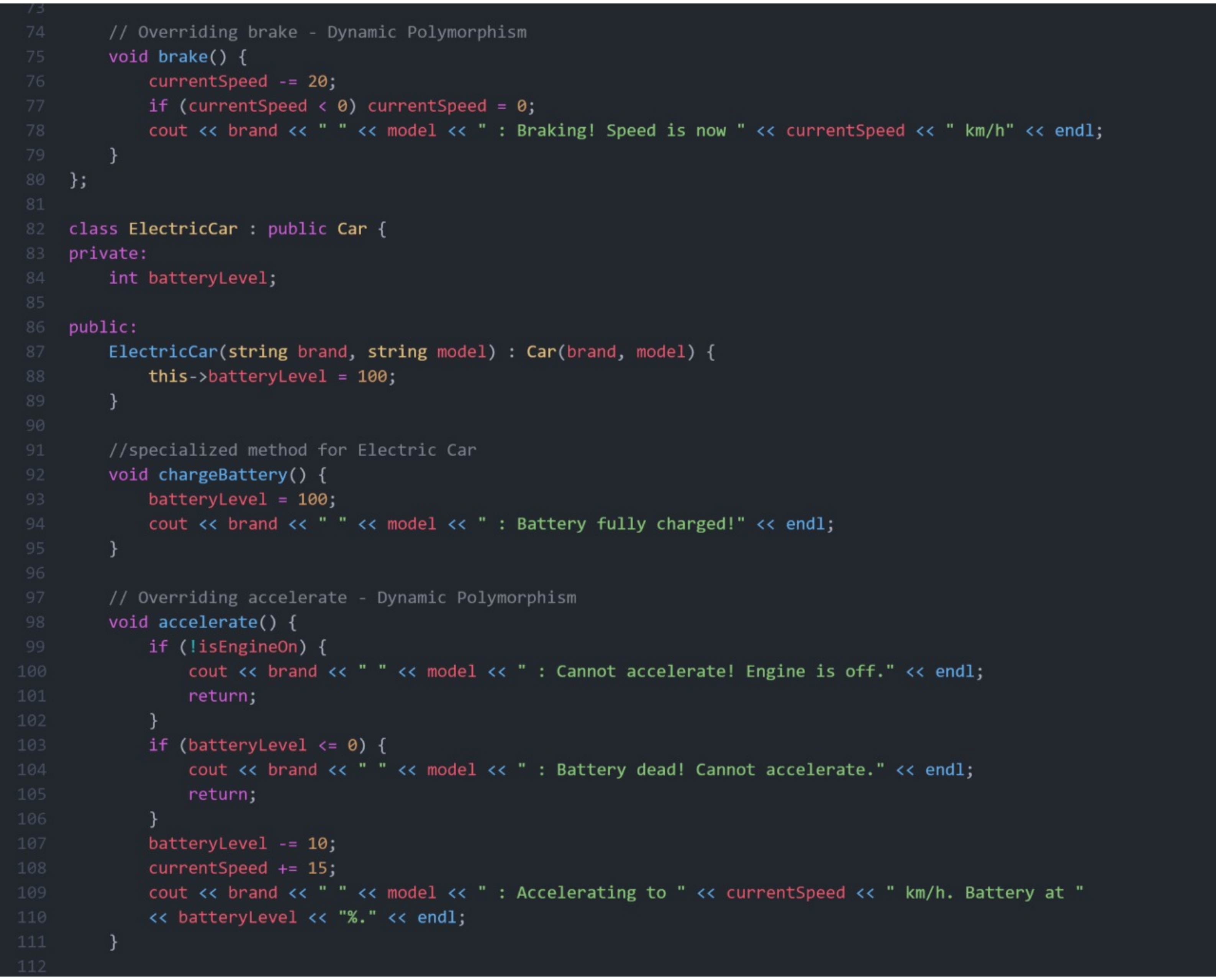
When we make object of manual car and electric car they will give different output if we call accelerate function



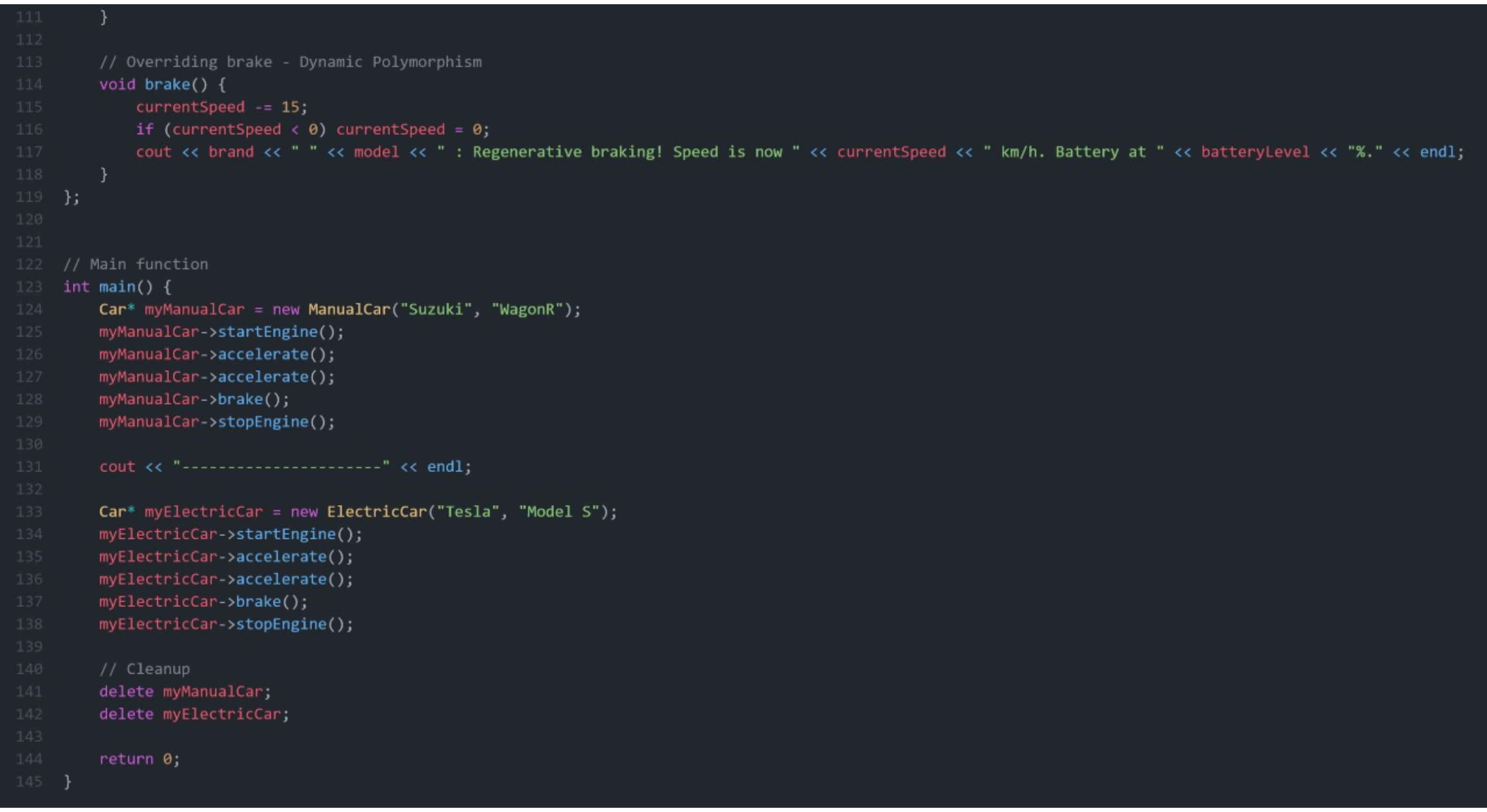
```cpp
#include <iostream>
#include <string>

using namespace std;

/*
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 behaviours 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.
*/
class Car {
protected:
    string brand;
    string model;
    bool isEngineOn;
    int currentSpeed;

public:
    Car(string brand, string model) {
        this->brand = brand;
        this->model = model;
        this->isEngineOn = false;
        this->currentSpeed = 0;
    }

    //Common methods for All cars.
    void startEngine() {
        isEngineOn = true;
        cout << brand << " " << model << " : Engine started." << endl;
    }
```



```cpp

    void stopEngine() {
        isEngineOn = false;
        currentSpeed = 0;
        cout << brand << " " << model << " : Engine turned off." << endl;
    }

    virtual void accelerate() = 0;  // Abstract method for Dynamic Polymorphism
    virtual void brake() = 0;       // Abstract method for Dynamic Polymorphism
    virtual ~Car() {}               // Virtual destructor
};

class ManualCar : public Car {
private:
    int currentGear;

public:
    ManualCar(string brand, string model) : Car(brand, model) {
        this->currentGear = 0;
    }

    //Specialized method for Manual Car
    void shiftGear(int gear) {
        currentGear = gear;
        cout << brand << " " << model << " : Shifted to gear " << currentGear << endl;
    }

    // Overriding accelerate - Dynamic Polymorphism
    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += 20;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }
```



```cpp
    // Overriding brake - Dynamic Polymorphism
    void brake() {
        currentSpeed -= 20;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Braking! Speed is now " << currentSpeed << " km/h" << endl;
    }
};

class ElectricCar : public Car {
private:
    int batteryLevel;

public:
    ElectricCar(string brand, string model) : Car(brand, model) {
        this->batteryLevel = 100;
    }

    //specialized method for Electric Car
    void chargeBattery() {
        batteryLevel = 100;
        cout << brand << " " << model << " : Battery fully charged!" << endl;
    }

    // Overriding accelerate - Dynamic Polymorphism
    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        if (batteryLevel <= 0) {
            cout << brand << " " << model << " : Battery dead! Cannot accelerate." << endl;
            return;
        }
        batteryLevel -= 10;
        currentSpeed += 15;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h. Battery at "
        << batteryLevel << "%." << endl;
    }

```



```cpp
    }

    // Overriding brake - Dynamic Polymorphism
    void brake() {
        currentSpeed -= 15;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Regenerative braking! Speed is now " << currentSpeed << " km/h. Battery at " << batteryLevel << "%." << endl;
    }
};


// Main function
int main() {
    Car* myManualCar = new ManualCar("Suzuki", "WagonR");
    myManualCar->startEngine();
    myManualCar->accelerate();
    myManualCar->accelerate();
    myManualCar->brake();
    myManualCar->stopEngine();

    cout << "----------------------" << endl;

    Car* myElectricCar = new ElectricCar("Tesla", "Model S");
    myElectricCar->startEngine();
    myElectricCar->accelerate();
    myElectricCar->accelerate();
    myElectricCar->brake();
    myElectricCar->stopEngine();

    // Cleanup
    delete myManualCar;
    delete myElectricCar;

    return 0;
}
```

Static Polymorphism : Same methods
different parameters

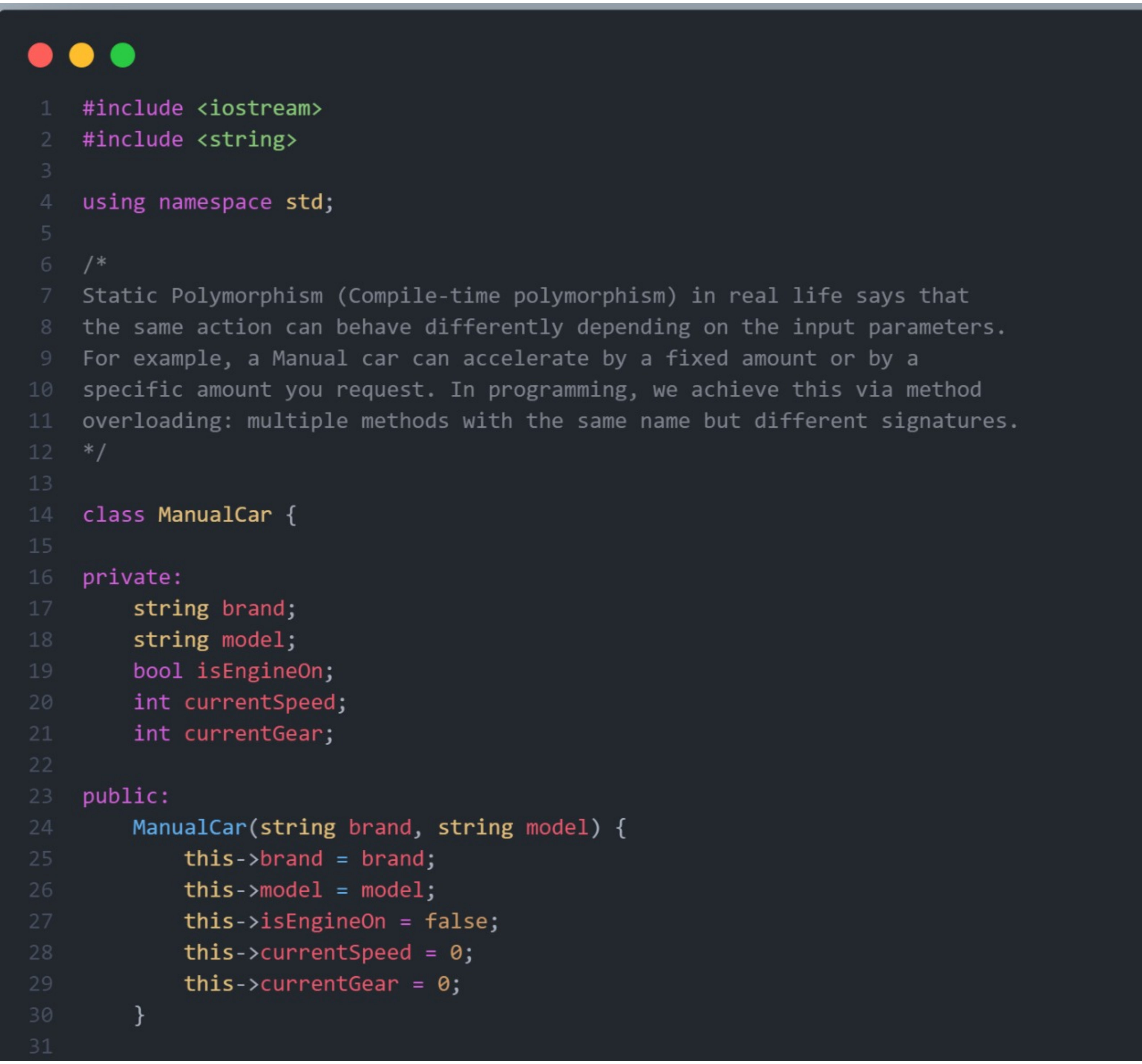
Human
↓
run()
↳ IsInDanger ()
↳ True
↳ False

Car Manual
↳ accelerate ()
↳ accelerate (int speed)
↓
how much
we push paddle
then accelerate
according to that

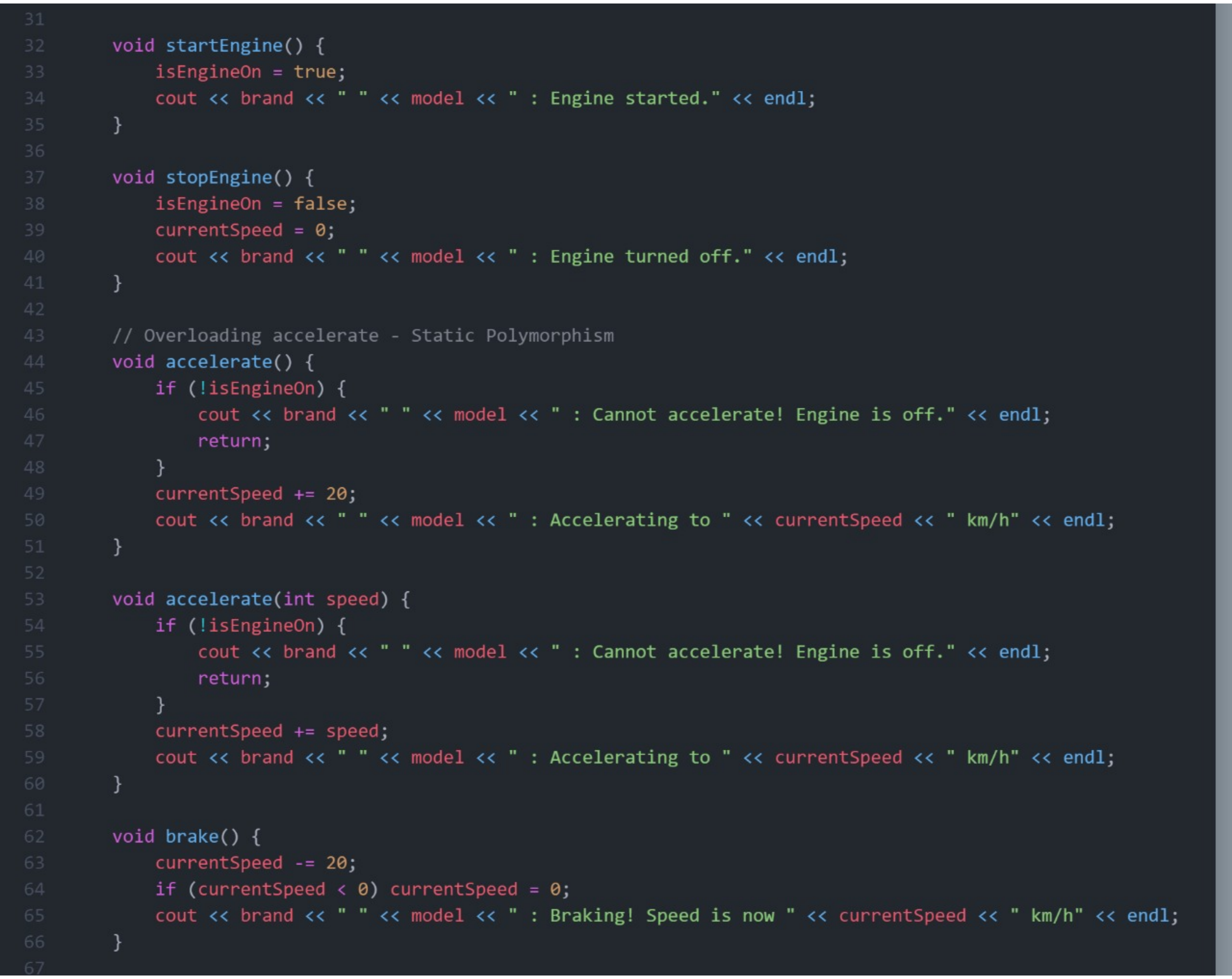
{ ↳ Default

Human has one method run
we pass (IsInDanger) as argument
if it is True then it will run
fast if it is false then it run
slow.

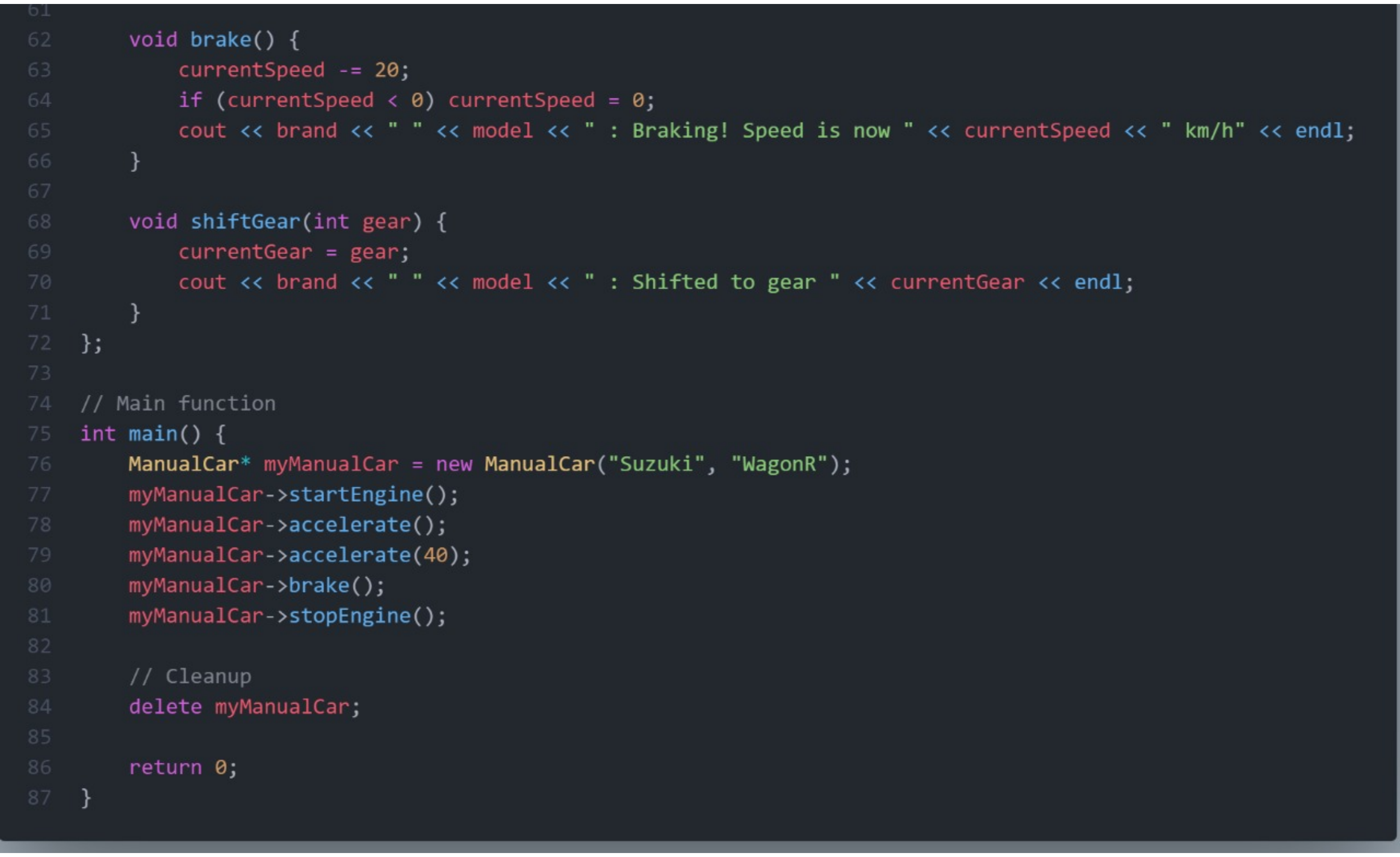
Both are same
method but
parameter is different



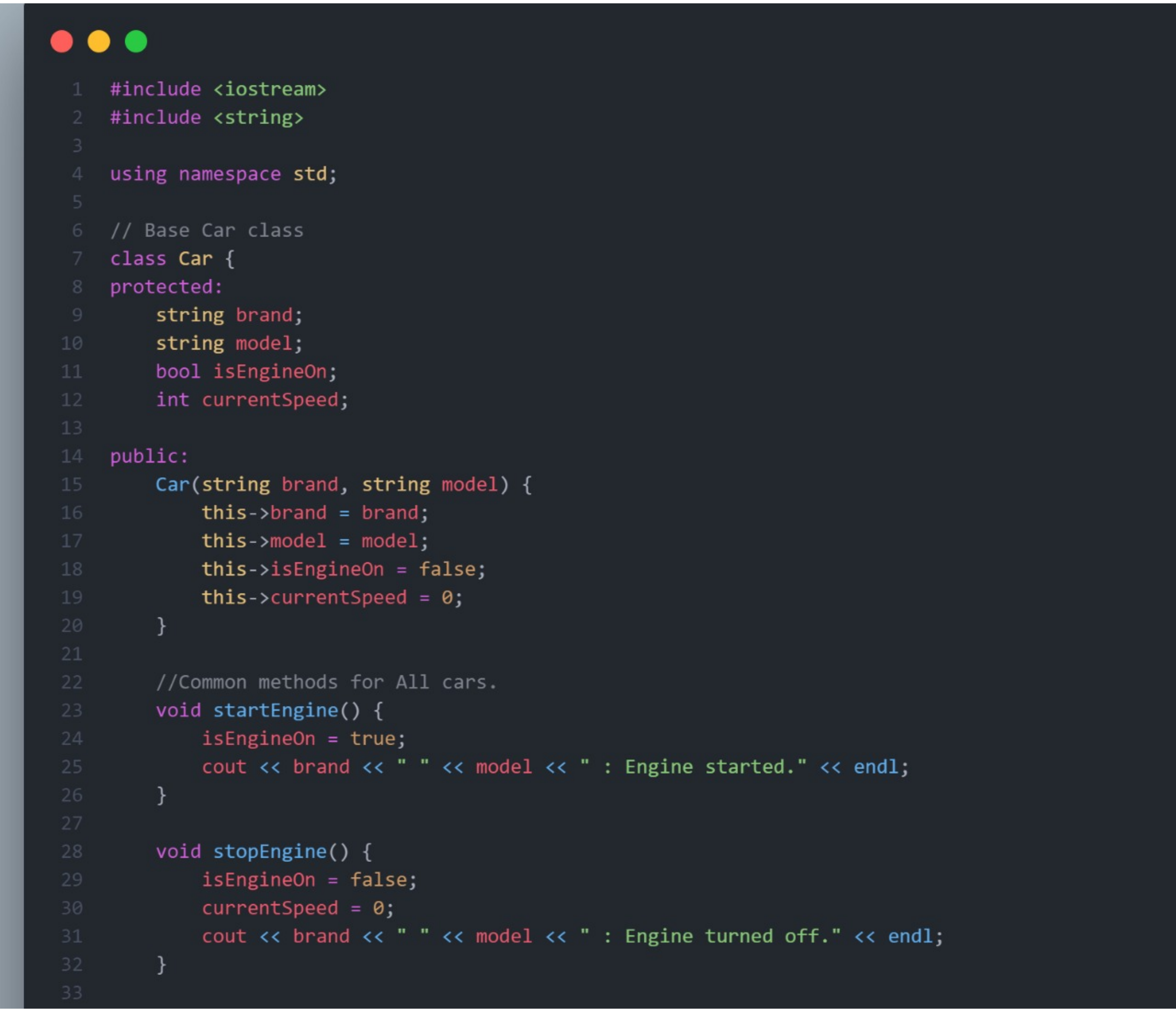
```cpp
#include <iostream>
#include <string>

using namespace std;

/*
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 fixed amount or by a
specific amount you request. In programming, we achieve this via method
overloading: multiple methods with the same name but different signatures.
*/

class ManualCar {

private:
    string brand;
    string model;
    bool isEngineOn;
    int currentSpeed;
    int currentGear;

public:
    ManualCar(string brand, string model) {
        this->brand = brand;
        this->model = model;
        this->isEngineOn = false;
        this->currentSpeed = 0;
        this->currentGear = 0;
    }

```



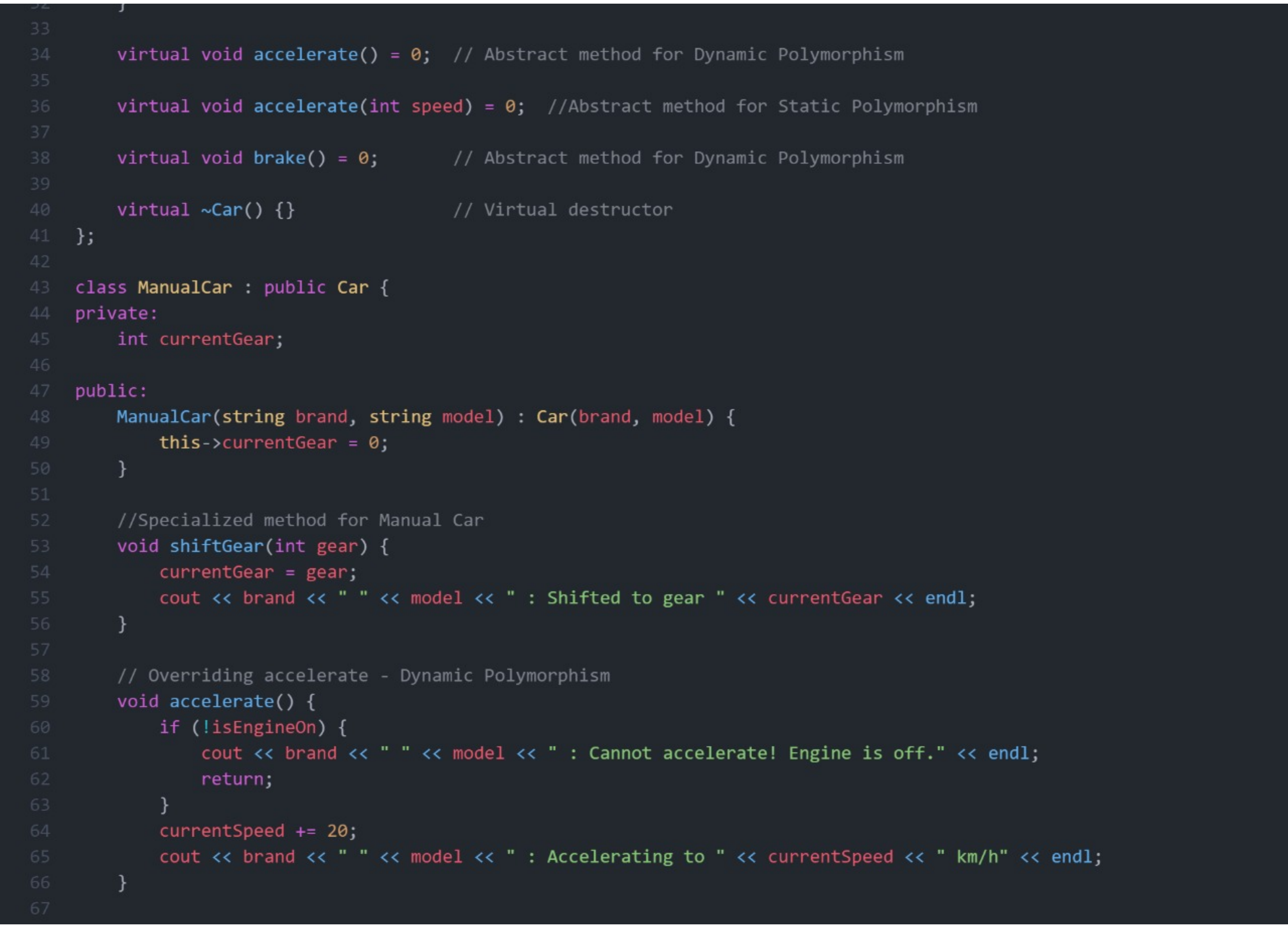
```cpp

    void startEngine() {
        isEngineOn = true;
        cout << brand << " " << model << " : Engine started." << endl;
    }

    void stopEngine() {
        isEngineOn = false;
        currentSpeed = 0;
        cout << brand << " " << model << " : Engine turned off." << endl;
    }

    // Overloading accelerate - Static Polymorphism
    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += 20;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }

    void accelerate(int speed) {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += speed;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }

    void brake() {
        currentSpeed -= 20;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Braking! Speed is now " << currentSpeed << " km/h" << endl;
    }
```



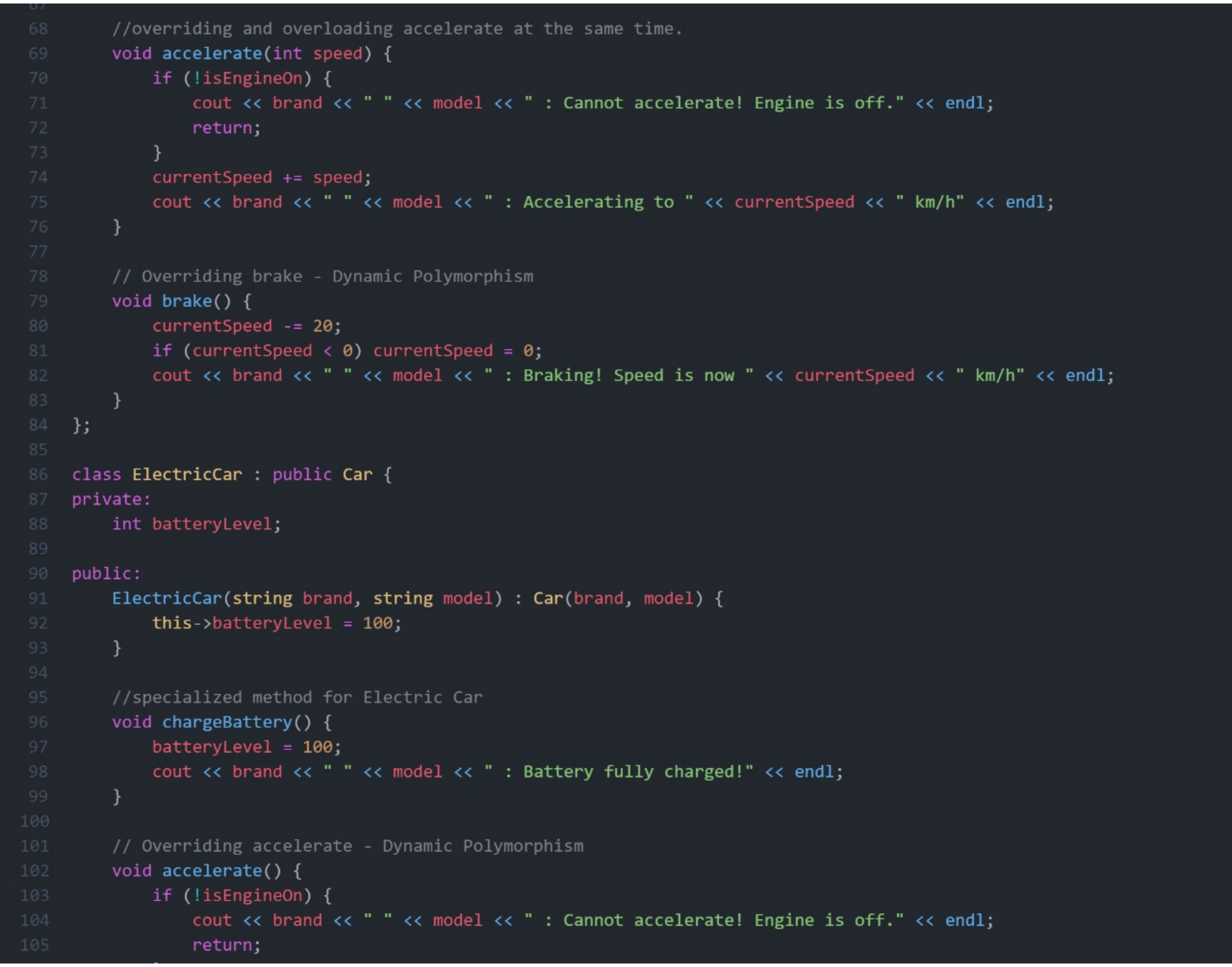
```cpp
    void brake() {
        currentSpeed -= 20;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Braking! Speed is now " << currentSpeed << " km/h" << endl;
    }

    void shiftGear(int gear) {
        currentGear = gear;
        cout << brand << " " << model << " : Shifted to gear " << currentGear << endl;
    }
};

// Main function
int main() {
    ManualCar* myManualCar = new ManualCar("Suzuki", "WagonR");
    myManualCar->startEngine();
    myManualCar->accelerate();
    myManualCar->accelerate(40);
    myManualCar->brake();
    myManualCar->stopEngine();

    // Cleanup
    delete myManualCar;

    return 0;
}
```



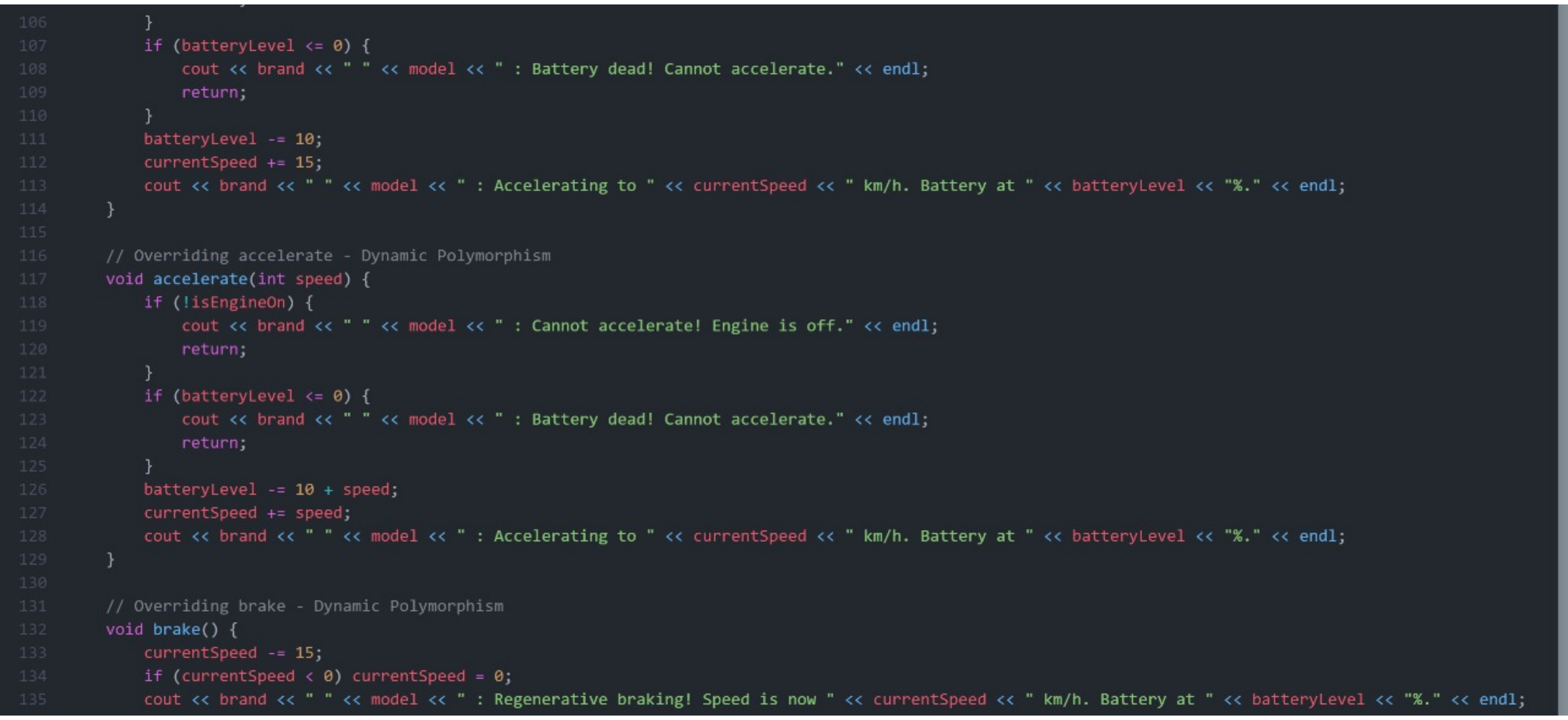
```cpp
#include <iostream>
#include <string>

using namespace std;

// Base Car class
class Car {
protected:
    string brand;
    string model;
    bool isEngineOn;
    int currentSpeed;

public:
    Car(string brand, string model) {
        this->brand = brand;
        this->model = model;
        this->isEngineOn = false;
        this->currentSpeed = 0;
    }

    //Common methods for All cars.
    void startEngine() {
        isEngineOn = true;
        cout << brand << " " << model << " : Engine started." << endl;
    }

    void stopEngine() {
        isEngineOn = false;
        currentSpeed = 0;
        cout << brand << " " << model << " : Engine turned off." << endl;
    }

```



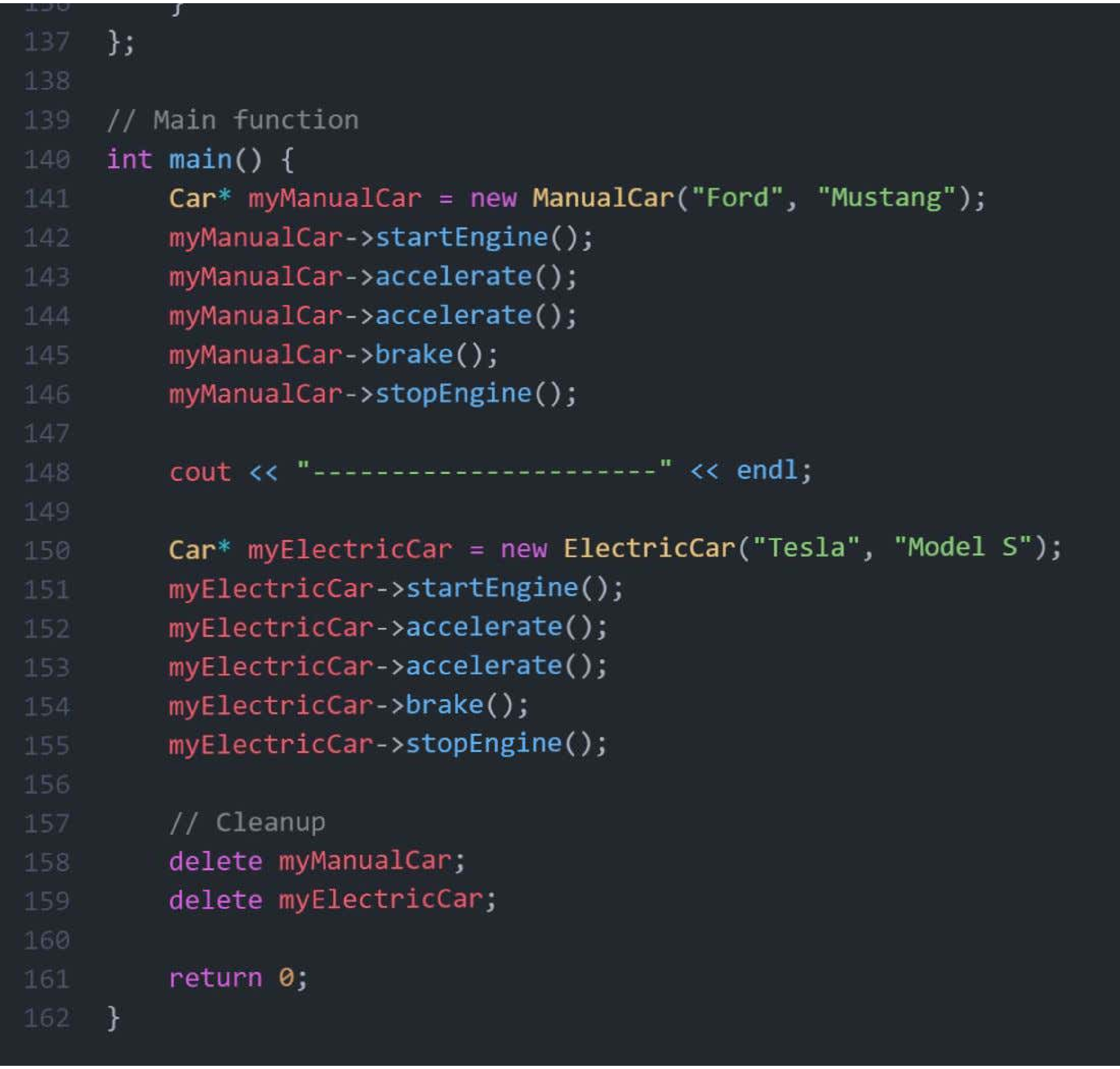
```cpp
    }

    virtual void accelerate() = 0;  // Abstract method for Dynamic Polymorphism

    virtual void accelerate(int speed) = 0;  //Abstract method for Static Polymorphism

    virtual void brake() = 0;       // Abstract method for Dynamic Polymorphism

    virtual ~Car() {}               // Virtual destructor
};

class ManualCar : public Car {
private:
    int currentGear;

public:
    ManualCar(string brand, string model) : Car(brand, model) {
        this->currentGear = 0;
    }

    //Specialized method for Manual Car
    void shiftGear(int gear) {
        currentGear = gear;
        cout << brand << " " << model << " : Shifted to gear " << currentGear << endl;
    }

    // Overriding accelerate - Dynamic Polymorphism
    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += 20;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }

```



```cpp
    //overriding and overloading accelerate at the same time.
    void accelerate(int speed) {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        currentSpeed += speed;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h" << endl;
    }

    // Overriding brake - Dynamic Polymorphism
    void brake() {
        currentSpeed -= 20;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Braking! Speed is now " << currentSpeed << " km/h" << endl;
    }
};

class ElectricCar : public Car {
private:
    int batteryLevel;

public:
    ElectricCar(string brand, string model) : Car(brand, model) {
        this->batteryLevel = 100;
    }

    //specialized method for Electric Car
    void chargeBattery() {
        batteryLevel = 100;
        cout << brand << " " << model << " : Battery fully charged!" << endl;
    }

    // Overriding accelerate - Dynamic Polymorphism
    void accelerate() {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
```



```
        }
        if (batteryLevel <= 0) {
            cout << brand << " " << model << " : Battery dead! Cannot accelerate." << endl;
            return;
        }
        batteryLevel -= 10;
        currentSpeed += 15;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h. Battery at " << batteryLevel << "%." << endl;
    }

    // Overriding accelerate - Dynamic Polymorphism
    void accelerate(int speed) {
        if (!isEngineOn) {
            cout << brand << " " << model << " : Cannot accelerate! Engine is off." << endl;
            return;
        }
        if (batteryLevel <= 0) {
            cout << brand << " " << model << " : Battery dead! Cannot accelerate." << endl;
            return;
        }
        batteryLevel -= 10 + speed;
        currentSpeed += speed;
        cout << brand << " " << model << " : Accelerating to " << currentSpeed << " km/h. Battery at " << batteryLevel << "%." << endl;
    }

    // Overriding brake - Dynamic Polymorphism
    void brake() {
        currentSpeed -= 15;
        if (currentSpeed < 0) currentSpeed = 0;
        cout << brand << " " << model << " : Regenerative braking! Speed is now " << currentSpeed << " km/h. Battery at " << batteryLevel << "%." << endl;
```



```cpp
    }
};

// Main function
int main() {
    Car* myManualCar = new ManualCar("Ford", "Mustang");
    myManualCar->startEngine();
    myManualCar->accelerate();
    myManualCar->accelerate();
    myManualCar->brake();
    myManualCar->stopEngine();

    cout << "----------------------" << endl;

    Car* myElectricCar = new ElectricCar("Tesla", "Model S");
    myElectricCar->startEngine();
    myElectricCar->accelerate();
    myElectricCar->accelerate();
    myElectricCar->brake();
    myElectricCar->stopEngine();

    // Cleanup
    delete myManualCar;
    delete myElectricCar;

    return 0;
}
```