Classes

Top-Down Approach

The concept of a "Blueprint" – Abstraction

- We have a blueprint of a Tesla car → create a Tesla Class
- We can now create/manufacture Tesla cars! → with Constructors
  - Create a default Constructor: `Tesla( );`

    " Model 3, white exterior, non-dual motor, non-full self driving"

  - Create Parameterized Constructor(s):
    " Which model? What color? Dual Motor? Full Self Driving? Etc.."

Polymorphism!

```
Tesla(string model, string ext_color);
Tesla(string model, string ext_color, bool dual_motor);
```

Model 3

White

Blue, Dual Motor

Model Y:

Red

Blue, Self Driving

Black, Dual Motor
Self Driving

The concept of a blueprint

Model 3

White

```
Tesla t1;
```

Default Constructor!

Blue, Dual Motor

```
Tesla t2(“Model3”, ”blue”, true)
```

Model Y:

Red

```
Tesla t3(”ModelY”, ”red”)
```

Blue, Self Driving

```
Tesla t4(“ModelY”, ”blue”, true)
```

Black, Dual Motor
Self Driving

```
Tesla t5(“ModelY”, “black”, true, true)
```

Parameterized Constructor

Data Members (Private)

- Think of what data members our Tesla class might have based on our constructors.

```
Tesla(string model, string ext_color, bool dual_motor, bool full_self_driving);
```

  - string _model
  - string _ext_color;
  - bool _dual_motor;
  - bool _full_self_driving;

- Data members are kept safe(private) -- Encapsulated

Cannot be directly accessed outside of the Tesla class!

Then how do we interact with the objects'(tesla cars) data? We need a Public Interface

We need a Public Interface

Getters (Accessors) & Setters(Mutators)

- Getter(Accessor): a member function that query a data member of the object and returns the value to the user.

- Setter(Mutator): member functions that modify the data members
  - Set a data member / attribute to a given value
  - Clear out a data member value

```
string getModel() const;
string getExteriorColor() const;
bool getIsDualMotor() const;
bool getIsFullSelfDriving() const;
```

```
void setModel(string model);
void setExteriorColor(string ext_color);
void setIsDualMotor(bool dual_motor);
void setIsFullSelfDriving(bool full_self_driving);
```

Encapsulation and Public Interface

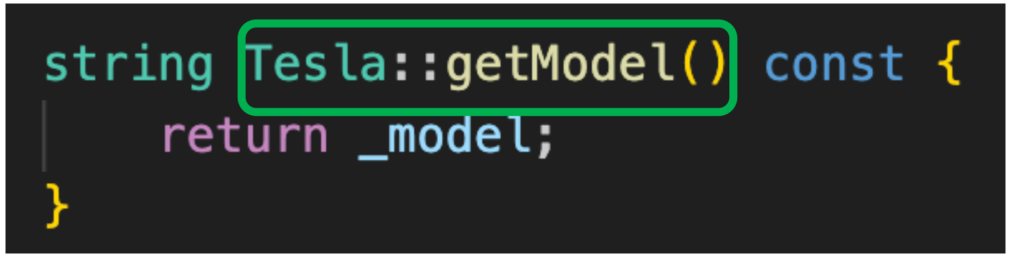
- public:
    - accessible outside the class definition
    - member functions
- private:
    - not accessible outside the class definition
    - data members

Encapsulation - Objects provide a public interface, while hiding the implementation details internally.

Put it Together!

- Header file (Tesla.h)
  - Provide the class definitions
  - Header Guards (`#ifndef TESLA_H`)
  - Order of variables(data members) and functions(member functions) is not important!

- Implementation file (Tesla.cpp)
  - Include header file(`#include "Tesla.h"`)
  - Provide the Implementation of Constructors, Getters, and Setters
  - Specify the scope using the scope resolution operator (::)



```
string Tesla::getModel() const {
    return _model;
}
```

Put it Together!

- Driver file (driver.cpp)
  - Has the main( ) function
  - This will be used to test your Tesla class implementation