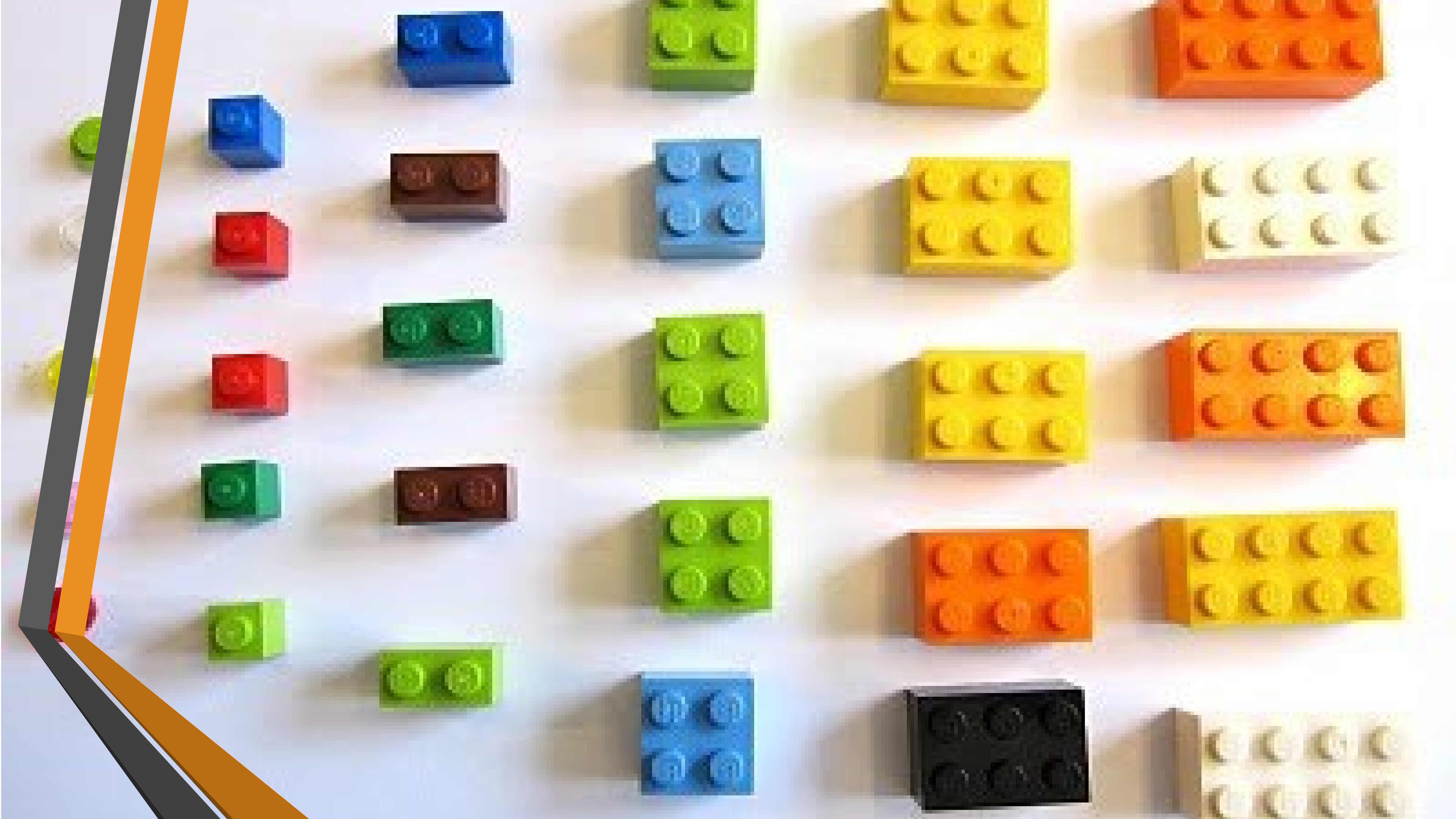
Object-Oriented Programming

Classes



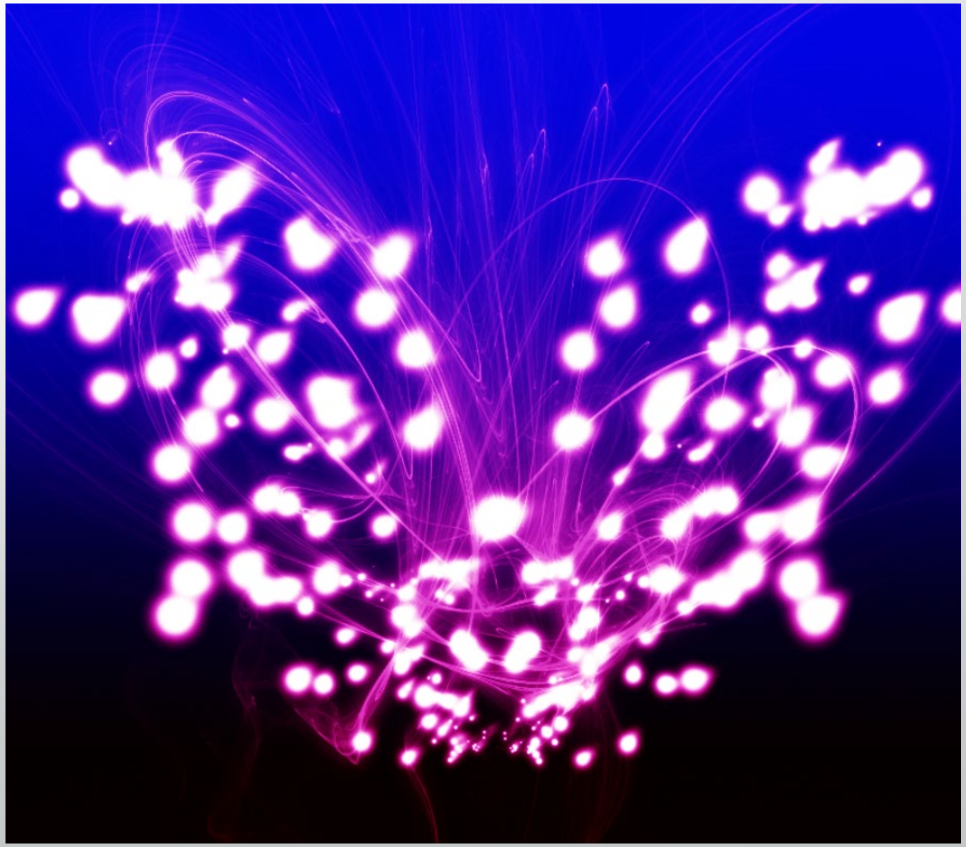
The Four Pillars of OOP

- **Abstraction**
- **Encapsulation**
- **Inheritance**
- **Polymorphism**

OOP - Abstraction

- **Abstraction:** Ability to define objects that represent abstract entities that do work, change state, and interact with other entities
- It is about **grouping** certain **behaviors** (methods) and **properties** (data) of an object that **defines** the object abstractly

What do you see?



OOP - Encapsulation

- **Encapsulation**: the grouping of data with the behaviors (methods) that do something with them
- **HIDES:**
  the data
  how it works
- The internal workings need to be hidden from the user of the class. **WHY**?
  - To protect the instances from being modified in a way that would make it invalid

What's inside? IDK

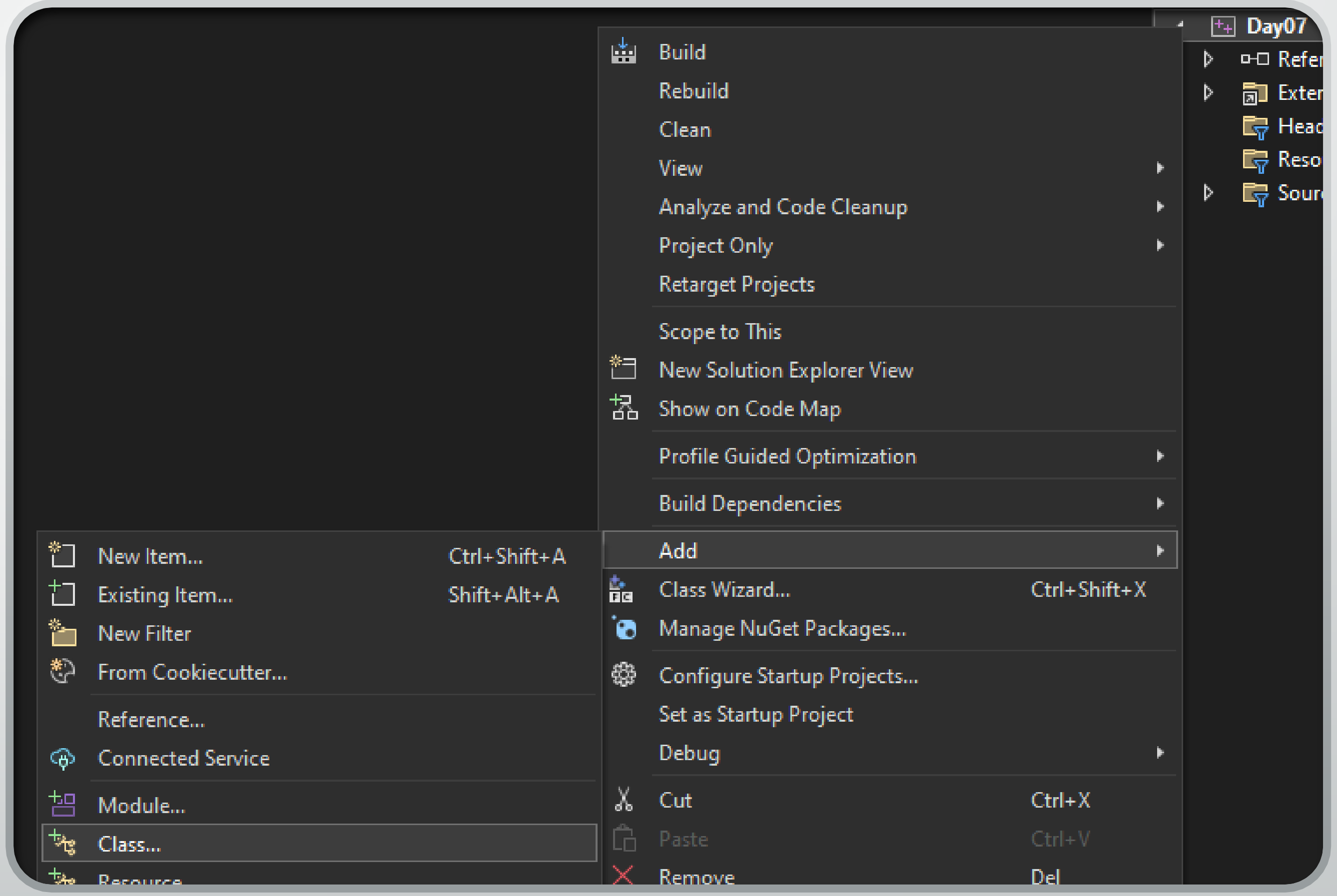
Object-Oriented Programming

Creating a Class

Creating a C++ Class

- You should not put everything in the same file.
- **Good programming**: put each class in their own files (.cpp, .h)

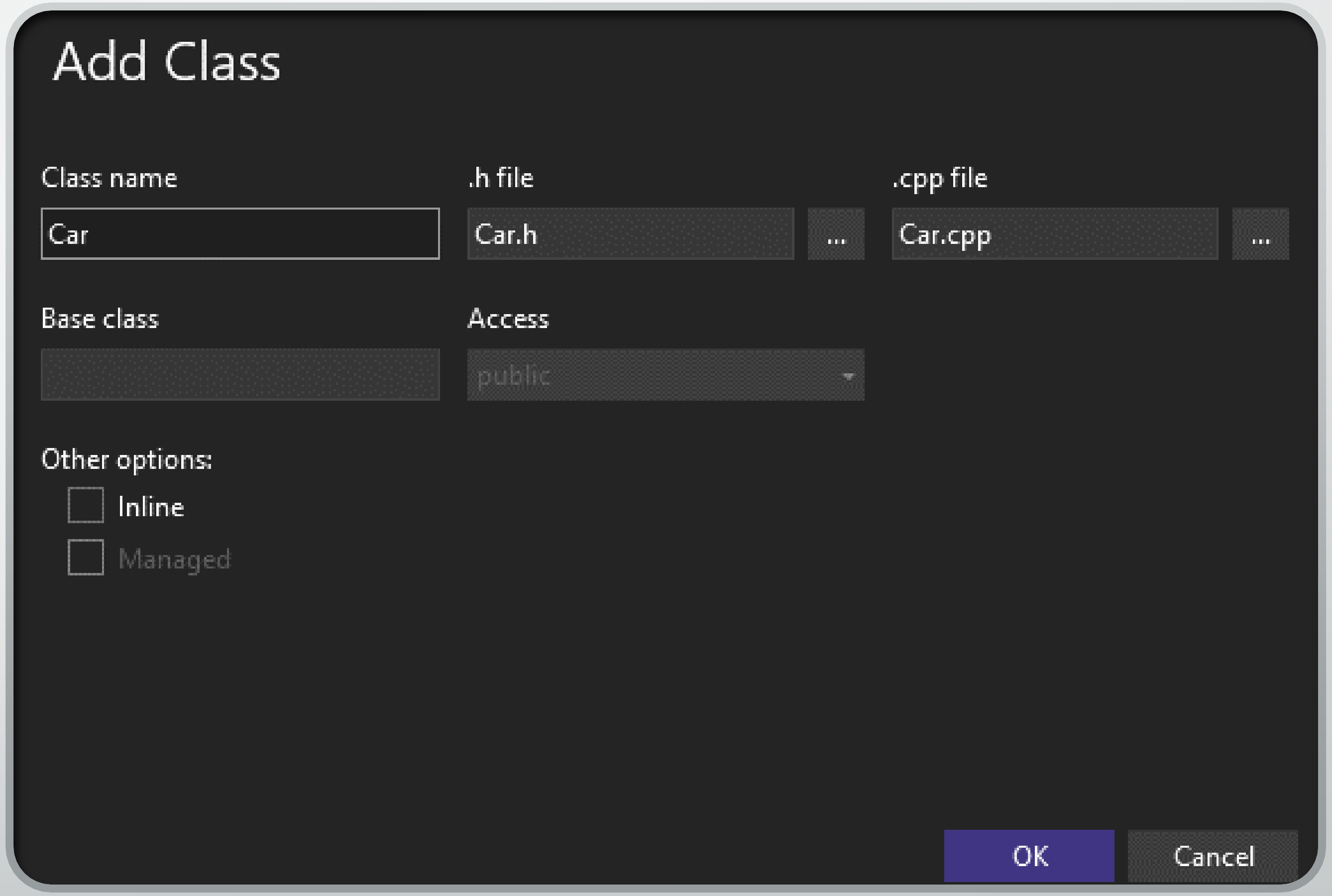
1. Right-click the project name in the Solution Explorer.
2. Select "Add->Class…" from the context menu.
3. In the dialog, enter the name of the class.



Creating a C++ Class

- Give the class a name that is descriptive.
- Start the class name with a capital letter.
- Use Pascal casing if the name is a compound word. EX: BattleCruiser.

- NOTE:
- Visual Studio will create 2 files for your class: a **.cpp** (code) and a **.h** (declarations).


Add Class

Class name: Car
.h file: Car.h
.cpp file: Car.cpp
Base class:
Access: public

Other options:
- [ ] Inline
- [ ] Managed

Object-Oriented Programming

Access Modifiers

Class Member Access Modifiers

- Access Modifiers determine the accessibility of the class members.
- **If not specified, the members are private.**

- **public**: ALL code can access it
- **private**: ONLY the current class can access it
- **protected**: the current class and any subclass (derived class)

Object-Oriented Programming

Fields

C++ Fields

- Fields are the data for your class (aka, the **data members**).
- Normally, you want to **hide** your data from the users of your class. This keeps them from modifying your data without your knowledge. You would do this by making the field private or protected.

C++ Fields

- A word on naming conventions…
- There are a LOT of different ways to name your fields.
- EX: m_firstName, _firstName, mFirstName, first_name, etc.
- Rules for naming:
  - Follow a standard industry convention
  - **The local naming convention in your company should override any industry convention.**

C++ Fields

```
class Car
{
private:
    int mModelYear;
    std::string mModel;
};
```

Object-Oriented Programming
Methods

C++ Methods

- Methods are the behavior for your class (aka, the **member functions**).

- The method **declaration** is provided in the **.h** header file.
- The method **definition** is provided in the **.cpp** source file.

C++ Methods

- The method **declaration** is provided in the **.h** header file.

```
class Car
{
private:
    int mModelYear;
    std::string mModel;
    std::string mMake;

public:
    std::string vehicleInformation();
};
```

C++ Methods

- The method **definition** is provided in the **.cpp** source file.

```
std::string Car::vehicleInformation()
{
    return std::to_string(mModelYear) + " " + mMake + " " + mModel;
}
```

Object-Oriented Programming

Getter/Setter Methods

Getters/Setters

- The getter/setter methods are special kind of methods that provide access to the data members (fields) of a class. They are the **gatekeepers** of the fields.
- You generally do not want to make your fields public to protected them.
- But you do need to usually provide some kind of “controlled” access.

Getters

- Getters (or accessors) return the value of a field.

```
int modelYear() const  //const says the method can’t modify anything
{
    return mModelYear;  //provides access to the field's value
}
```

Setters

- Setters (or mutators) allow callers to give fields a new value.

```
void modelYear(int modelYear)
{
    mModelYear = modelYear;
}
```

Object-Oriented Programming

Constructors

C++ Constructors

- **Constructors** are special methods that are used to initialize the data members of a class.
- Constructors (C++) | Microsoft Learn
- RULES:
  - Has the same name as the class.
  - CANNOT have a return value (not even void).
  - You can have as many overloaded constructors as you want.
  - They can have any access modifier (although public is usually used).

C++ Constructors
Member Initialization lists

- A constructor can optionally have a member initialization list.
- Member initialization lists initializes the class members *before* the constructor body.
- **Prefer** member initialization lists over assigning values to members in the constructor body.

C++ Constructors
Member Initialization lists

```
Hero(std::string heroName) : mName(heroName)
{
    //constructor body
}
```

C++ Constructors
Default Constructors

- **Default Constructors** have no parameters, but they can have parameters with default values.
- **Default Constructors** are one of the special member functions. If no constructors are declared in a class, the compiler provides an implicit inline default constructor.

C++ Constructors

Default Constructors

```
** no parameters
Car()
{ }


** all parameters have default values
Car(int modelYear=1908, std::string make = "Ford", std::string model = "A")
    :mModelYear(modelYear), mMake(make), mModel(model)
{}
```

Object-Oriented Programming
Structs

Structs

- A struct is a user-defined composite type. Which means, it can have multiple fields or members that have different types.
- Structs preceded classes.
- In C++, structs are the same as classes *except* that members are public by default.
- Structs are usually used for light-weight data objects like a color or rect.

Structs

```
struct COLOR
{
    unsigned short red;
    unsigned short green;
    unsigned short blue;
    unsigned short alpha;
};
```