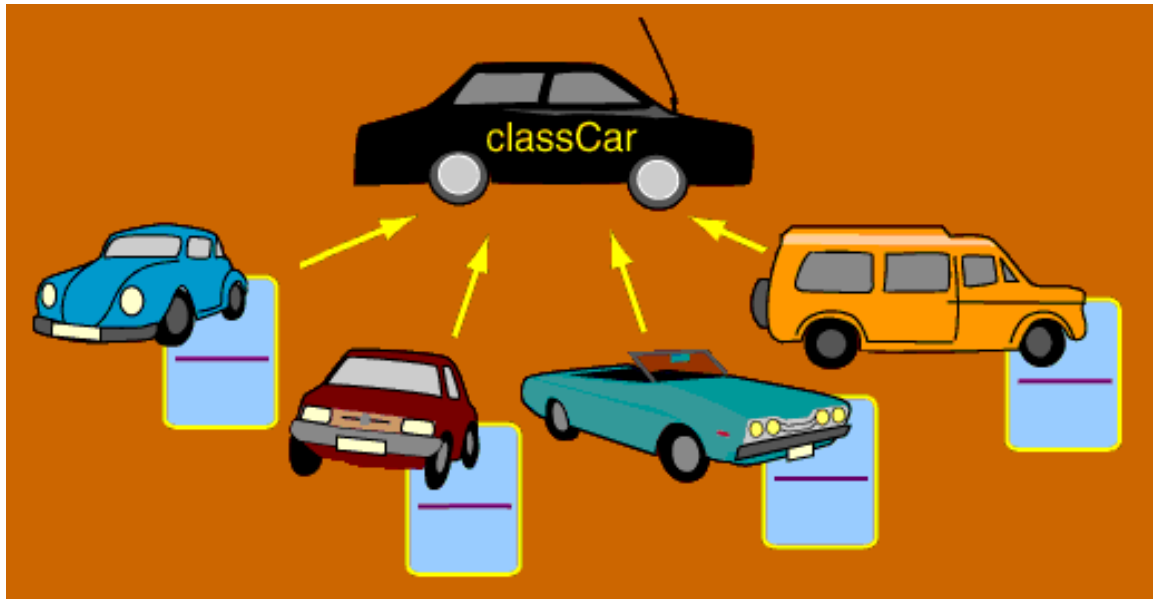
Object Orientation



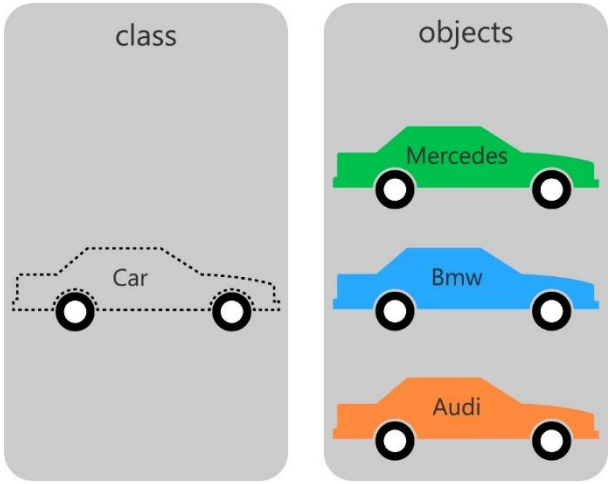
View the World as Classes of Objects

- An “object” is the computer model of a real object
- In object oriented programming, we group the objects our program is concerned about into classes
- Everything object in a class shares:
  - How it is described – what data is used to describe it
  - How it can be manipulated – what actions can be performed on any object in the class

Example Class: Car

How Cars are Described

- Make
- Model
- Year
- Color
- Owner
- Location
- Mileage



Actions that can be applied to cars

- Create a new car
- Transfer ownership
- Move to a new location
- Repaint
- Delete a car

Description == List of data fields

- Make - string
- Model - string
- Year - integer
- Color - enum
- Owner - string
- Location – gps: longitude/latitude in degrees/minutes/seconds
- Mileage - float

C vs. C++ : Object Oriented Data

In C

- Class is similar to a structure definition
- Fields are elements of the structure
- An object is a pointer to an instance of the structure
- Create a data type for objects in a class with a typedef

In C++

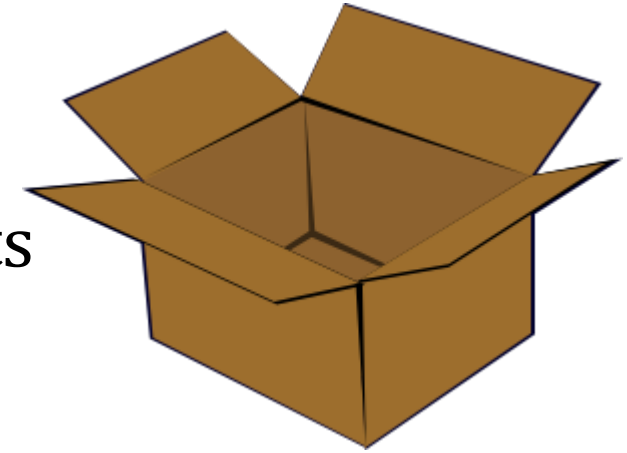
- Classes are built in to the language
- Fields are variables declared in the class
- An object is an instance of a class
- Classes are by definition a user defined data type

Example C "Car" class

```
struct car_s {
        char * make;
        char * model;
        int year;
        enum colors color;
        char *owner
        gps_coord location;
        float mileage;
};
```

```
typedef struct car_s *car;
```

Object Oriented Encapsulation



- Concept: Leave dealing with cars up to the car experts
- If you aren't a car expert, don't go under the hood!
- Make one place the "auto-mechanic" place
- That place has the only code that modifies car objects!
- If any one outside of that place wants to interact with cars, it has to invoke a service provided by the expert

Encapsulation in C

- Put all the car related stuff in a single file
  - “car.c”
- Put the user interface stuff for cars in a single file
  - “car.h”
- Every access to car stuff outside of “car.c” must be through function calls
- All car functions:
  - Declared in “car.h” (so they can be invoked after #include “car.h”)
  - Defined/Implemented in “car.c”

What kinds of functions work on cars?

- Definitely need one or more "creator" functions
  - car makeCar(char *make,char *model,char *year);
- And if you can create it… you'd better be able to delete it
  - void deleteCar(car c);
- "getters" and "setters"
  - char * getCarMake(car c);
  - void setCarColor(car c,enum colors newColor);
- Any actions you want to perform on cars
  - void transferCarOwnership(car c,char *newOwner);

What would a creator look like?

```
car makeCar(char *make,char *model,char *year) {
        car c=(car)malloc(sizeof(struct car_s));
        c->make=make;
        c->model=model;
        c->year=year;
        c->color=white;
        c->owner=NULL;
        c->location=getLocation(factory);
        c->mileage=0.0;
        return c;
}
```

Getters and Setters

```
char * getCarMake(car c) { return c->make; }


void setCarColor(car c,enum colors newColor) {
        c->color=newColor; }

…
```

Using an Object in C

```
#include “car.h”

car myCar=makeCar(“VW”,”Passat”,2010);
setCarColor(myCar,white);
transferCarOwnership(myCar,“Tom Bartenstein”);
moveCar(myCar,getLocation(“East Gym Parking Lot”));
```

Objects as Parameters

- Notice that every function that works on cars (except the creator) takes a car object pointer as it's first argument.
- That's so we know WHICH car to work on

- In C++ there is a new notation…
  - myCar->setColor(white)
  - under the covers, passes "mycar" into the setColor method
  - under the covers, in setColor, the "this" variable refers to the object being worked on

Class Inheritance

- All race cars are cars, but race cars are a special sub-class of cars
  - Anything you can do with a car, you can also do with a race car
  - There are things you can do with race cars you can't do with cars
    - I want to be able to use all car functions on race cars
    - I want some new functions that I need only for race cars
  - Any data used to describe a car is also needed to describe a race car
  - I may have new data to describe a race car that I don't need for all cars

Example C "racecar" class

```
struct rcar_s {
        char * make;
        char * model;
        int year;
        enum colors color;
        char *owner
        gps_coord location;
        float mileage;
        float topSpeed;
};
```

```
typedef struct rcar_s *racecar;
```

Inheritance in C++

- Won't go into details at this point, but inheritance is even easier in C++ than in C.

```
class racecar : car  {
        float topSpeed;
        …
}
```

Why Object Orientation?

- Imposes structure on design
  - Forces everything into an object/action way of thinking
  - Reduces the number of choices we need to consider
- Establishes Responsibility / Traceability
  - If the "car" data structure has the wrong data, there is a bug in the "car.c" file… it can't be anywhere else!
- Establishes Areas of Expertise
  - Go to the car mechanic to get our car fixed… she knows how to fix it
- Re-Use
  - I can use the same classes in different programs

Object Orientation vs. C++/Java

- Object orientation is not a language… it's a way of thinking!
- It is possible to code object orients code in C
- It is possible to code structurally (non-object oriented) in C++ or Java
- C++ and Java provide features to make object oriented programming easier
- These extra features also make C++ and Java more complicated than C

Resources

- Programming in C, No references.
- Wikipedia Object Oriented Programming https://en.wikipedia.org/wiki/Object-oriented_programming
- Wikipedia Inheritance https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
- Object Oriented Programming Tutorials
  - http://www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm
  - https://docs.oracle.com/javase/tutorial/java/concepts/