What is Inheritance?

In this lesson, we'll be learning about the core concept of the object-oriented paradigm, i.e., Inheritance and why there is a need for it?

WE'LL COVER THE FOLLOWING

Why do We Need Inheritance?

In the classes chapter, we've covered the `HAS-A` relationship. We know a class `HAS-A` data members and member functions. Now, we want the data members, and member functions of the class are accessible from other classes. So, the capability of a class to derive properties and characteristics from another class is called `Inheritance`. In inheritance, we have `IS-A` relationship between classes e.g a *car* is a *vehicle* and a *ship* is a *vehicle*.

Let's take the example of `Vehicle` here.

Vehicle Class

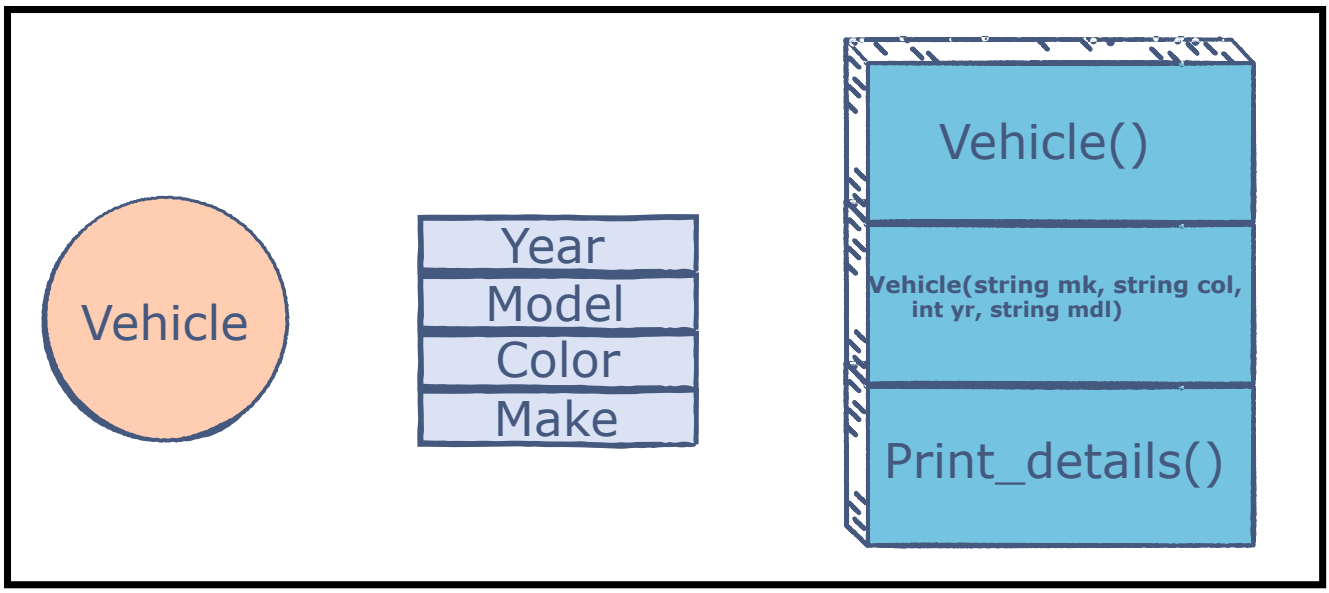
In a Vehicle class, we have many data members like *Make*, *Color*, *Year* and *Model*. A `Vehicle` *HAS-A* Model, Year, Color and Make.

Implementation of `Vehicle` Class

Let's look at the implementation of `Vehicle` class:

```
class Vehicle{
  protected:
  string Make;
  string Color;
  int Year;
  string Model;

  public:
  Vehicle(){
    Make = "";
    Color = "";
    Year = 0;
    Model = "";
  }

  Vehicle(string mk, string col, int yr, string mdl){
    Make = mk;
    Color = col;
    Year = yr;
    Model = mdl;
  }

  void print_details(){
    cout << "Manufacturer: " << Make << endl;
    cout << "Color: " << Color << endl;
    cout << "Year: " << Year << endl;
    cout << "Model: " << Model << endl;
  }
};

int main(){
  Vehicle v("Ford Australia", "Yellow", 2008, "Falcon");
  v.print_details();
}
```

The following illustration depicts the structure of the `Vehicle` class:



These attributes are also attributes of all *Cars*, *Ships* and *Airplanes* but every type of `vehicle` has some attributes that are different from other types of vehicles, as we will see in detail.

Cars Class

The implementation of a `Cars` class needs the same data members and member functions of `Vehicle` class but we have to include them in the `Cars` class. Cars do have a trunk and every trunk has a capacity to store things upto some limit.

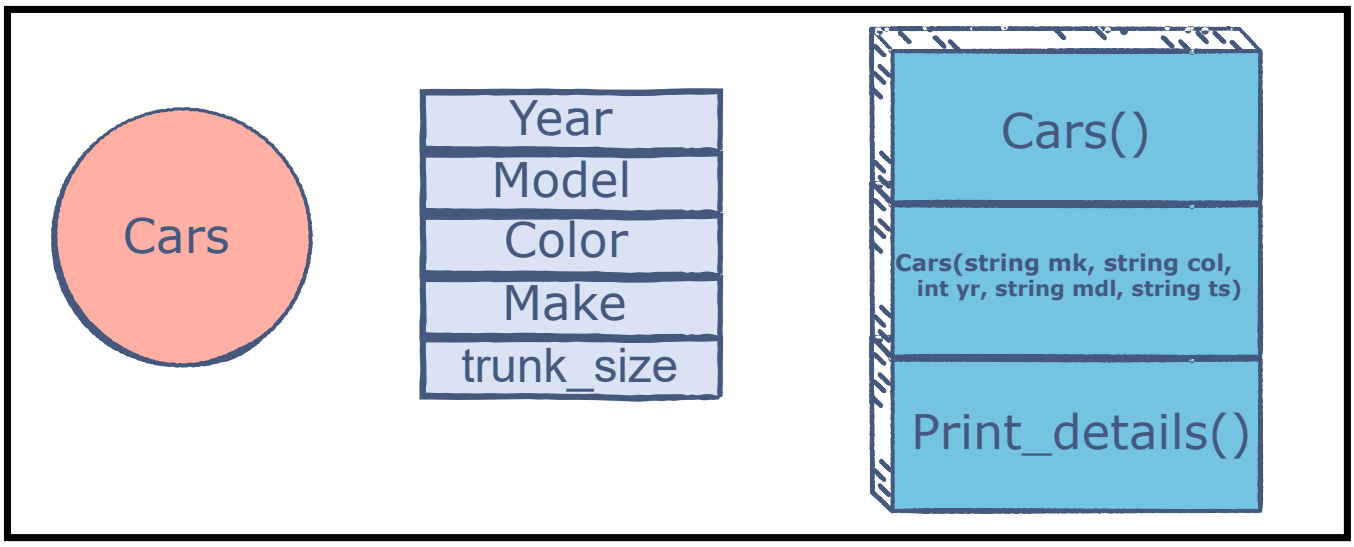
Implementation of `Cars` Class

Let's look at the implementation of the `Cars` class:

```
class Cars{
  string Make;
  string Color;
  int Year;
  string Model;
  string trunk_size;


  public:
  Cars(){
    Make = "";
    Color = "";
    Year = 0;
```

```
    Model = "";
    trunk_size = "";
  }

  Cars(string mk, string col, int yr, string mdl, string ts){
    Make = mk;
    Color = col;
    Year = yr;
    Model = mdl;
    trunk_size = ts;
  }

  void print_details(){
    cout << "Manufacturer: " << Make << endl;
    cout << "Color: " << Color << endl;
    cout << "Year: " << Year << endl;
    cout << "Model: " << Model << endl;
    cout << "Trunk size: " << trunk_size << endl;
  }
};

int main(){
  Cars car("Chevrolet", "Black", 2010, "Camaro", "9.1 cubic feet");
  car.print_details();
}
```

The following illustration depicts the structure of the `Cars` class:



Ships Class

The implementation of a `Ships` class needs the same data members and

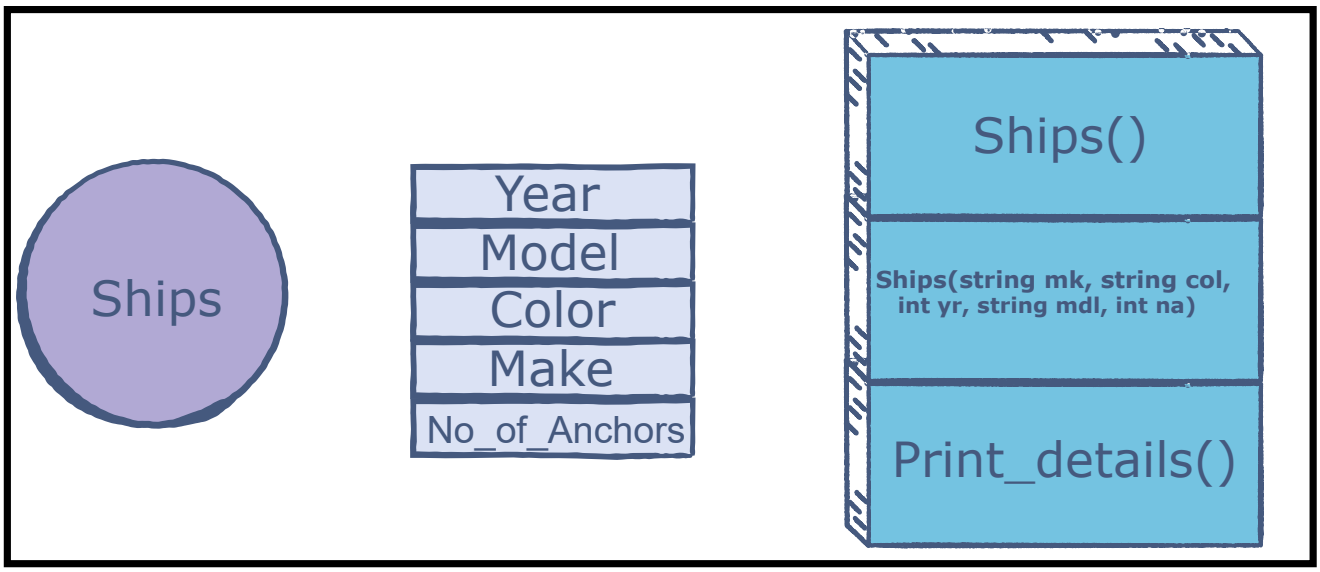
member functions of `Vehicle` class but we have to include them in the `Ships` class. Ships do have anchors and they vary in numbers.

Implementation of `Ships` Class

Let's look at the implementation of the `Ships` class:

```
class Ships{
  string Make;
  string Color;
  int Year;
  string Model;
  int Number_of_Anchors;


  public:
  Ships(){
    Make = "";
    Color = "";
    Year = 0;
    Model = "";
    Number_of_Anchors = 0;
  }

  Ships(string mk, string col, int yr, string mdl, int na){
    Make = mk;
    Color = col;
    Year = yr;
    Model = mdl;
    Number_of_Anchors = na;
  }

  void print_details(){
    cout << "Manufacturer: " << Make << endl;
    cout << "Color: " << Color << endl;
    cout << "Year: " << Year << endl;
    cout << "Model: " << Model << endl;
    cout << "Number of Anchors: " << Number_of_Anchors << endl;
  }
};

int main(){
  Ships ship("Harland and Wolff, Belfast", "Black and whilte",
             1912, "RMS Titanic", 3);
  ship.print_details();
}
```

The following illustration depicts the structure of the `Ships` class:



Class Name | Class Data Member | Class Member Functions

In the declared classes for different types of vehicles (`Cars` and `Ships`), we have many repetitive attributes which should be in one base class and should be inherited in the derived classes.

In the next lesson, we'll be learning about base class and derived class.