Classes and Objects:

In OOP we deal with objects, which belong to classes of objects having a similar characteristics (member variables and member functions).

In doing this we distinguish the interface to an object (inputs and outputs to the black box) and the implementation of an object (what's inside the black box).

Information Hiding Principles:

The user of a module (e.g., class) should know everything needed to use the module and nothing more.

The implementer of a module should know everything necessary to implement the module and nothing more.

Why? It's to keep the modules as independent as possible. As a user I cannot make use of whatever I might know or be able to find out about the implementation, and as implementer I cannot make use of whatever I might know or be able to find out about how it will be used. The interface is in effect a contract between the users and implementers. It's what we both can depend on.

This means I'm free to change either the implementation or the use of the module, so long as I don't change them interface. This aids maintenance.

In C++ structs and classes are the same except:
In structs by default all members are public.
In classes by default all members are private.
In both, you can explicitly declare anything to be either public or private. Many

programmers think that declaring things explicitly is the best style.

Suppose we want a function to search a deck d and find whether and where a card c is in the deck. There are several styles of interface:

Standalone function: find (c, d) or find (d, c)
Member function of decks: d.find(c) // in d find c
Member function of cards: c.find(d) // find c in d

It's purely a matter of style and what you think is easiest to use, most readable, and most understandable. I will follow the book, and make it a member function of Card.

Remember that in C++ things have to be declared before they are used. But:
The declaration of Deck obviously makes use of Card.
But, if we look closely, the declaration of Card, has to make of the Deck, because the declaration of find is:

```
int find (const Deck& d);
```

So we have circular declarations, which isn't allowed. C++ solves this with a forward declaration:

```
class Deck; // forward declaration of Deck

class Card {
….
int find (const Deck& d);
…
```

```
};

class Deck {
… uses of Card …
};
```