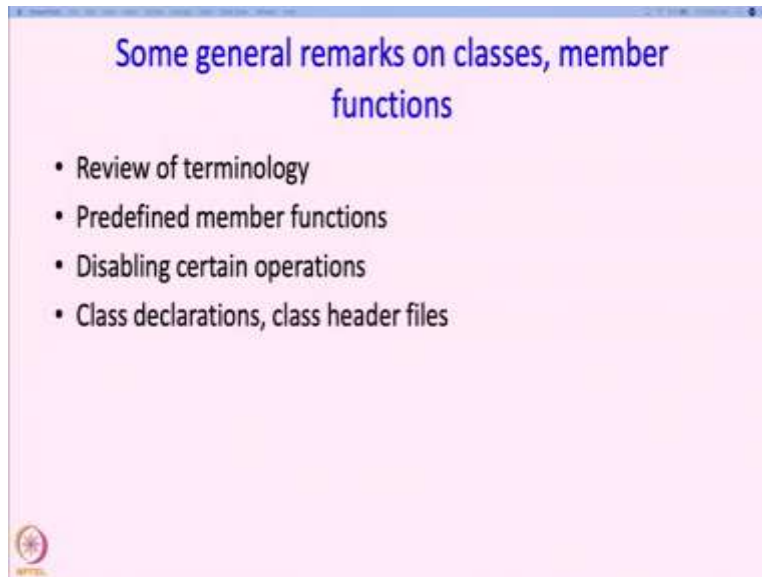


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 21 Part- 6
Classes
General Remarks

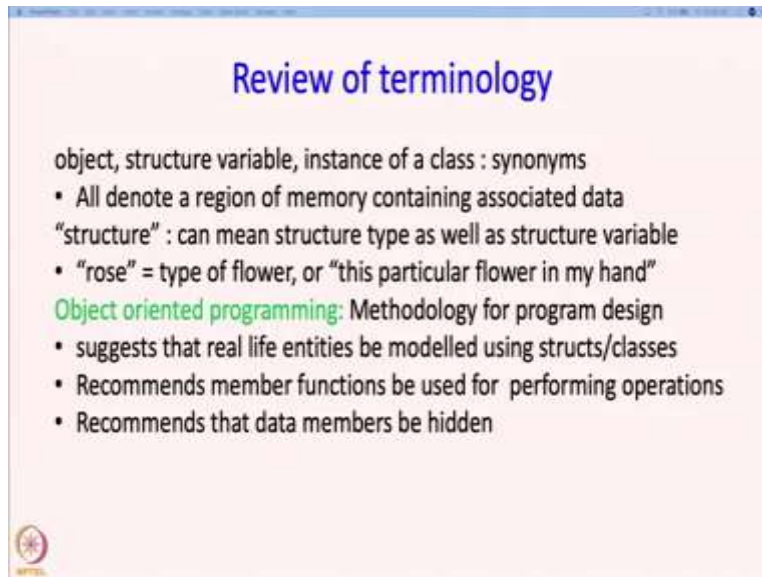
Welcome back. In the previous segment we discussed graphics classes, and input/output classes. In this segment I will make some general remarks and conclude this sequence of segments.

(Refer Slide Time: 0:29)



So, I will first do a review of terminology, then I will talk about predefined member functions, I will talk about disabling certain operations, and then I will talk about class declarations and class header files.

(Refer Slide Time: 0:44)



Review of terminology

object, structure variable, instance of a class : synonyms

- All denote a region of memory containing associated data

“structure” : can mean structure type as well as structure variable

- “rose” = type of flower, or “this particular flower in my hand”

Object oriented programming: Methodology for program design

- suggests that real life entities be modelled using structs/classes
- Recommends member functions be used for performing operations
- Recommends that data members be hidden

So, the terms object, structure variable, instance of a class, these are synonyms, and all of them denote a region of memory, and that is the region of memory which contains the data associated with that object, or structure variable, or the class instance. Now, the term structure can mean a structure type, as well as a structure variable and which one it is will be clear from the context.

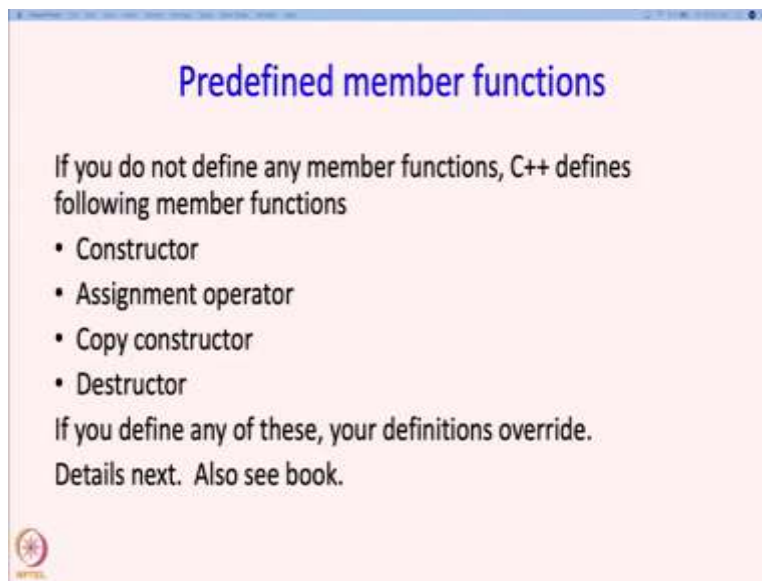
This is really not that confusing, because this kind of usage is present in day to day life. For example, when we talk about a rose, you might mean a type of flower or we might mean, this particular flower in my hand. So, our use of the term structure is similar, to this kind of usage. Then the term object oriented programming also gets used a lot.

Object oriented programming is a methodology for a program design. This methodology suggests that real life entities should be modelled using structures and classes. So, basically if your program is about, certain real life entities, this methodology says that you should make structures or classes corresponding to those real life objects. And, if you are going to operate upon these real-life entities, then you should make member functions. You should not directly access the data members. The data members should be hidden. The idea is that, data members can not be accessed without any discipline; you may just set the values of some data members. Instead of that if you use a function, then you can only do a certain fixed set of things, so by

providing an interface using functions, the designer says that look, this is how you should be using it and this is and if you use it in this way then your program is going to be correct.

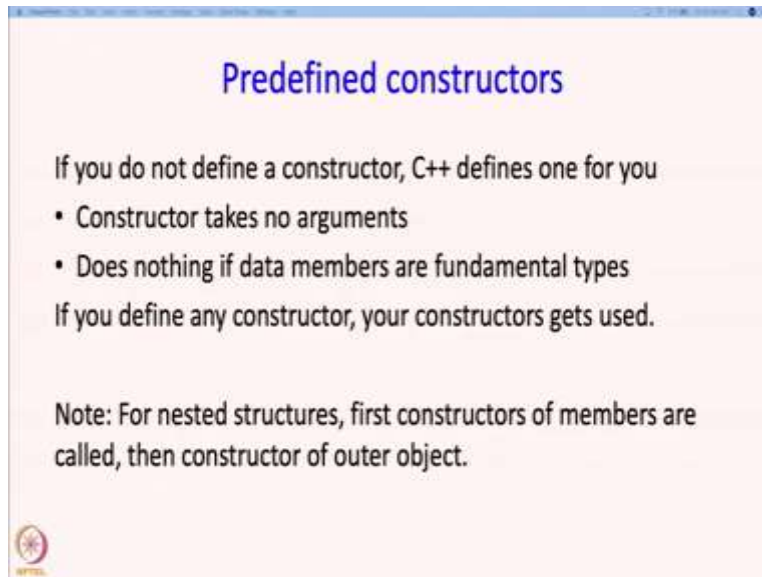
Now object oriented programming has many other features, so, for example, there is a notion of polymorphism, or there is a notion of inheritance. These are outside the scope of this course but they are discussed in the book.

(Refer Slide Time: 3:20)



Now in C++ many member functions are already defined for you. So if you do not define any member functions, C++ still has some member functions defined. For example, C++ does have a constructor defined for every class. The assignment operator is also likewise defined, and something called a copy constructor is defined, and something called a destructor is also defined. You can define any of these and we will see a little bit more detail, and when you do the definitions, your definitions will override the predefined member function definitions. So, I will talk about some details now, but really for a complete description you will need to read the book.

(Refer Slide Time: 4:16)



Predefined constructors

If you do not define a constructor, C++ defines one for you

- Constructor takes no arguments
- Does nothing if data members are fundamental types

If you define any constructor, your constructors gets used.

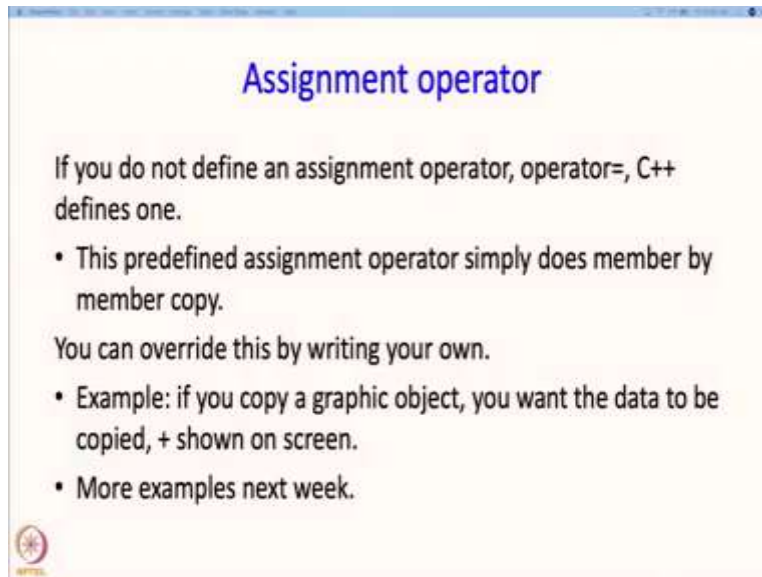
Note: For nested structures, first constructors of members are called, then constructor of outer object.

So, predefined constructors are always there, if you do not define a constructor. So this constructor takes no arguments. So, what does it do? Well it does nothing, so if the data members are fundamental types, it really does nothing. So, in fact, whenever you are defining a struct earlier and you are just saying struct P, then this predefined constructor was being called to construct that object P.

If you define any constructor, with arguments without arguments, then your constructors are going to get used, and the constructor, the predefined constructor is not going to be used. C++ is basically saying that look you are taking in your own hands, how an object should be constructed. So my predefined constructors will not, will not be used. I want to make a rather wider note, as we are discussing constructors.

So, if you have nested structures, so say you have a rectangle structure, a struct, and inside that there is a point struct. So in that case, so the constructors work in the following manner. So, first the point objects inside the rectangle will be constructed using the constructors for the point, and only after the data members are constructed will the constructor for the outer object get called.

(Refer Slide Time: 5:56)




Assignment operator

If you do not define an assignment operator, operator=, C++ defines one.

- This predefined assignment operator simply does member by member copy.

You can override this by writing your own.

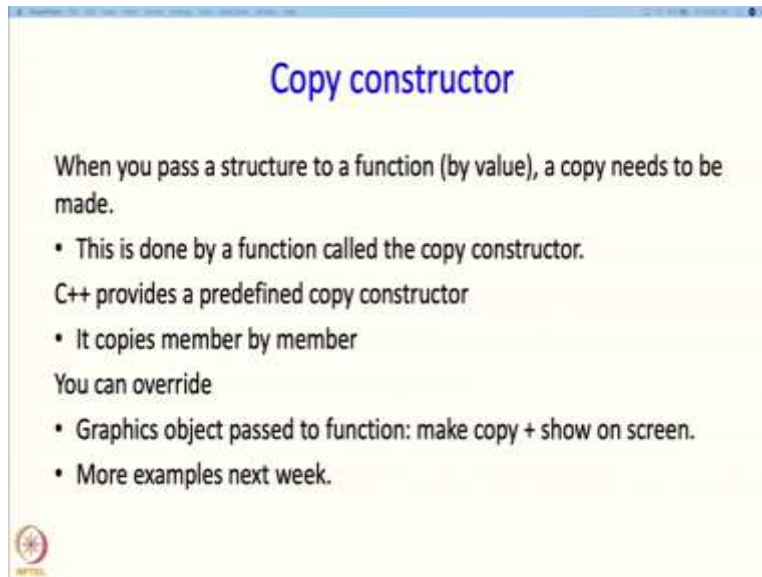
- Example: if you copy a graphic object, you want the data to be copied, + shown on screen.
- More examples next week.



When we talked about assignments for struct, and really you should think of that as being an assignment operator that C++ predefined for you. This predefined assignment operator simply does member by member copy. So, this is this is what you have been used before we began this discussion of member functions. When we talked about structs, well you can assign one struct to another struct, and that was just doing member by member copy.

You can override this, by writing your own assignment operator. Just as we define the plus operator, you can define the operator equal to. Why is this necessary? Well here is one example, if you copy, if you make an assignment to a graphics object. You want the data to be copied, plus the changes, because of the assignment need to also be shown on the screen. So, your assignment operator can, in fact, make the changes to the screen just as you are doing the assignment. And we will see more examples of this next week.

(Refer Slide Time: 7:15)



Copy constructor

When you pass a structure to a function (by value), a copy needs to be made.

- This is done by a function called the copy constructor.

C++ provides a predefined copy constructor

- It copies member by member

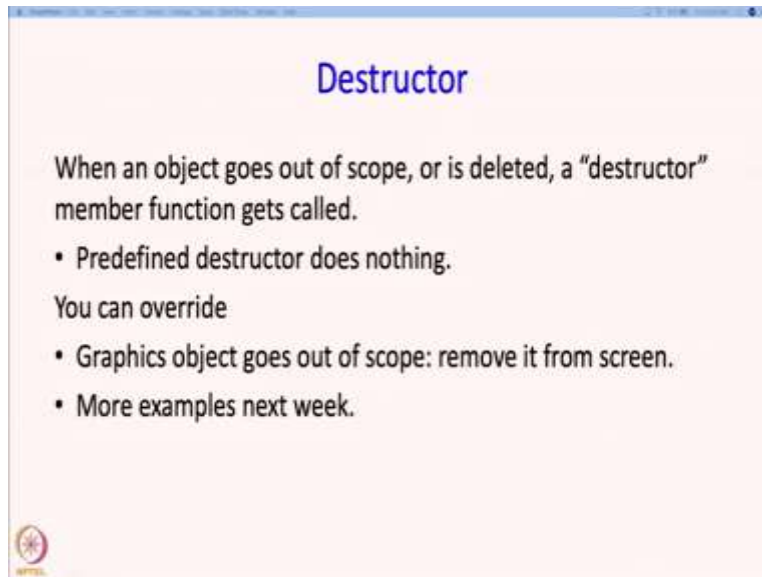
You can override

- Graphics object passed to function: make copy + show on screen.
- More examples next week.

A copy constructor is something that is used when you pass a structure to a function. So, when you pass a structure to a function by value, we have discussed that a copy needs to be made. Now this is done by a function called the copy constructor. So this copy constructor gets called to make that copy. C++ provides a predefined copy constructor. So, what does the predefined copy constructor do? Well it does a member by member copy.

So the new object that is created in the activation frame, of the called function will be a copy, will be a member by member copy. You can override this, and again for graphics. You may need not only to make a copy, but you may need to change something on the screen. So, if you move the object subsequently. You will want this copy to move. So some additional things might need to be done in addition to just copying the data members. So this can be done, so this is why we have this thing called a copy constructor. And next week we will see more examples for the need of the copy constructor.

(Refer Slide Time: 8:34)



Destructor

When an object goes out of scope, or is deleted, a "destructor" member function gets called.

- Predefined destructor does nothing.

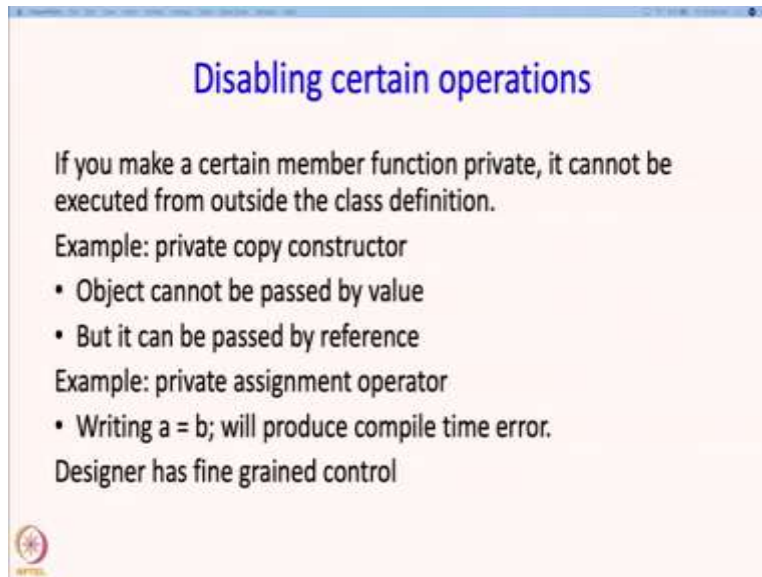
You can override

- Graphics object goes out of scope: remove it from screen.
- More examples next week.

A destructor sounds like really really dangerous term, but it is needed for the following case. So, when an object goes out of scope or is deleted by the delete operation, C++ actually calls a destructor member function. The predefined destructor does nothing. So, basically, there is the area of the object and nothing really happens, well actually that is not quite true. The destructors of the nested objects will also get called. So, that is what the predefined destructor does. Just like the predefined constructor calls the constructor of the nested objects, the predefined destructor would also call the destructor of the nested objects.

But, you can override. So, instead of doing nothing, you can do you can get something to be done. So, for example, if a graphics object is going out of scope, then you do not want to see it on the screen. So if you are designing the graph the graphics system, then you will put in the destructor the code to remove the object from the screen. And there are other cases, as well where destructors need to do something interesting, and we are going to see examples of that next week.

(Refer Slide Time: 10:01)



The slide is titled "Disabling certain operations" in blue text. It contains the following text:

If you make a certain member function private, it cannot be executed from outside the class definition.

Example: private copy constructor

- Object cannot be passed by value
- But it can be passed by reference

Example: private assignment operator

- Writing `a = b;` will produce compile time error.

Designer has fine grained control

There is a small logo in the bottom left corner of the slide.

Now, this whole framework of member functions is quite interesting, in that it allows you to disable certain operations. Basically, the idea is the following if you make certain member functions private, then those member functions cannot be executed from outside of the class definition. So basically, for practical purposes you are saying that look you are not allowed to do this operation.

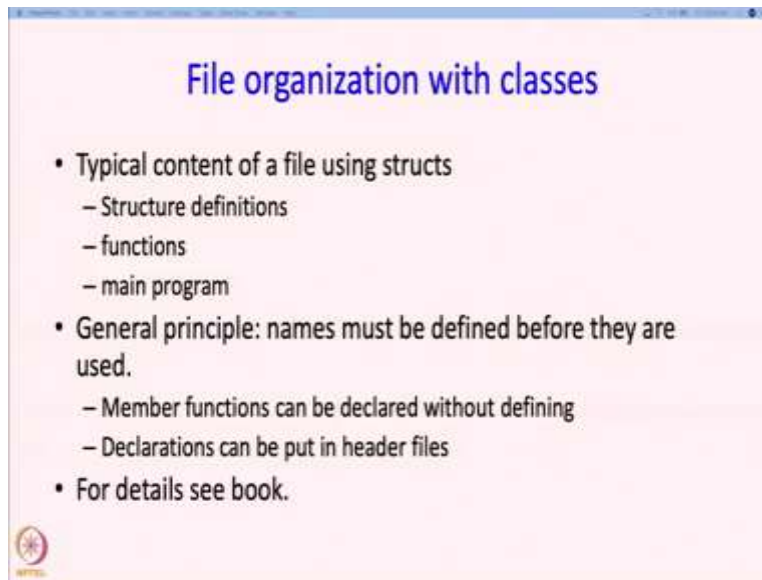
So, suppose you make the copy constructor private, then it cannot be called from outside which means that as a designer you are saying look, I do not want you to pass this object by a value to any function. You may want to do this; you may want to say that look I do not want a copy to be made of this object. But it can be passed by reference, so that is allowed. But a copy cannot be made. Similarly, you can make the assignment operator private.

In that case if you write something like `a=b;` you will be producing a compile time error, again you are really telling the user of that class please do not please do not make copies of this, I do not want you to make copies of this. So, we could have taken this approach, when we design the graphics objects. We could have said that you cannot make copies of the graphic objects.

So, that makes some sense, because after all when you make a copy you need to do some additional work and we might have said that look we have already done that additional work by

allowing you to create a new graphics object. That would have been a reasonable design, but I guess allowing for copies and doing that extra work is perhaps convenient to the user and therefore we did that. But C++ gives you the option of saying do not make a copy, so basically these these access control operators and member functions together give the designer great amount of fine control.

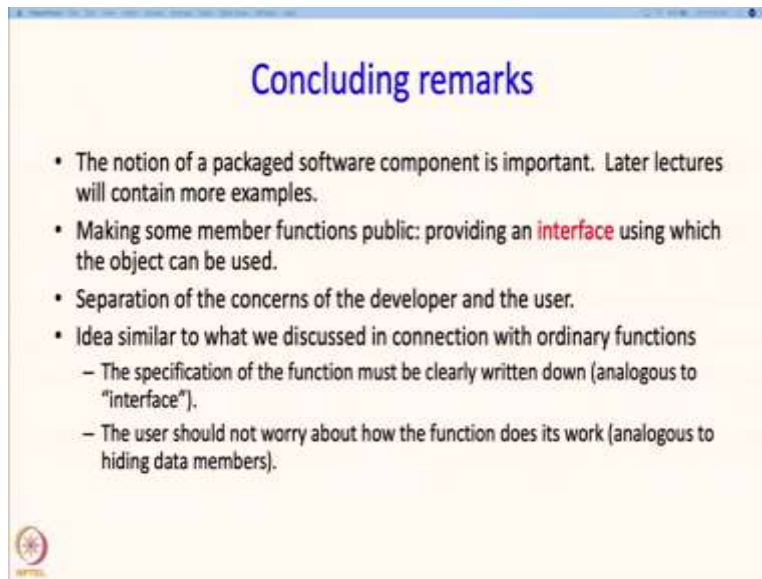
(Refer Slide Time: 12:11)



Now, I want to say a little bit about organizing files with classes. So typically if you have a program and if you have files in it the content will be something like this. So there will be structure definitions, there are functions; there will be the main program. The general principle is still the same; names must be defined before they are used. And, just as we said that functions can be defined, can be declared, without defining. I guess I should say the general principle is that names must be declared before they are used.

So, just as we said that ordinary functions can be declared without being defined, you can have member functions be declared without defining. And these declarations can also be put in header files. So, all this is reasonably consistent, reasonably similar to the way we talked about function header files for functions. And I am not going to discuss this in more detail, but it is discussed in good detail within the book.

(Refer Slide Time: 13:26)



The slide is titled "Concluding remarks" in a blue font. It contains a bulleted list of four main points, with the second point having two sub-points. The text is on a light yellow background. A small logo is visible in the bottom left corner of the slide.

- The notion of a packaged software component is important. Later lectures will contain more examples.
- Making some member functions public: providing an **interface** using which the object can be used.
- Separation of the concerns of the developer and the user.
- Idea similar to what we discussed in connection with ordinary functions
 - The specification of the function must be clearly written down (analogous to "interface").
 - The user should not worry about how the function does its work (analogous to hiding data members).

Now, I want to conclude this lecture sequence, this sequence of segments. I want to conclude this lecture and the point that I want to make, sort of the major high level point is that the notion of packaged software components is important, just as you have, you can have hardware components the idea over here is that you can have software components which are nicely packaged. And which are expected to be used in a certain manner and not in some other manners.

So, those, the way it is going to be used, is clearly indicated by the designer. And later lectures will contain more examples. Making some member functions public, is like providing an interface, sort of like a control panel. So, that is what the designer is telling you, "Please use this control panel or please use this set of member functions do not try to use the object in other ways."

So, basically, things are put behind this control panel and so the designer is saying well you do not need to worry about what goes on inside. So, I am telling you what these buttons are going to do, or I am going to tell I am going to tell you what these member functions are going to do, and I have tested my program and I have guaranteed that they do exactly what I am telling you.

And these ideas are really very similar to what we discussed in connection with ordinary functions. So, what we said there was the following, we said that the specification of the function

must be clearly written down and the specification of the function is really analogous to the interface, which must be written down as well and explained.

So, if you designed a class then for the benefit of the user, you should clearly explain what the member functions are doing. You do not have to explain to the user, how the implementation happens. But you should tell what are they really doing. So, for example, you can say, that move on a graphic subject will cause the object to be moved. How exactly that move gets implemented is not is not to be defined or is not to be discussed in this context.

Then the user should not worry about, how the function does its work, that is what we said and here, the idea is that the data members should be hidden and, in fact, the user should not concern, himself or herself with, how the implementation of the class happens. So, that is it that is the end of this lecture. As usual, please look at the exercises at the end of this chapter. Thank you.