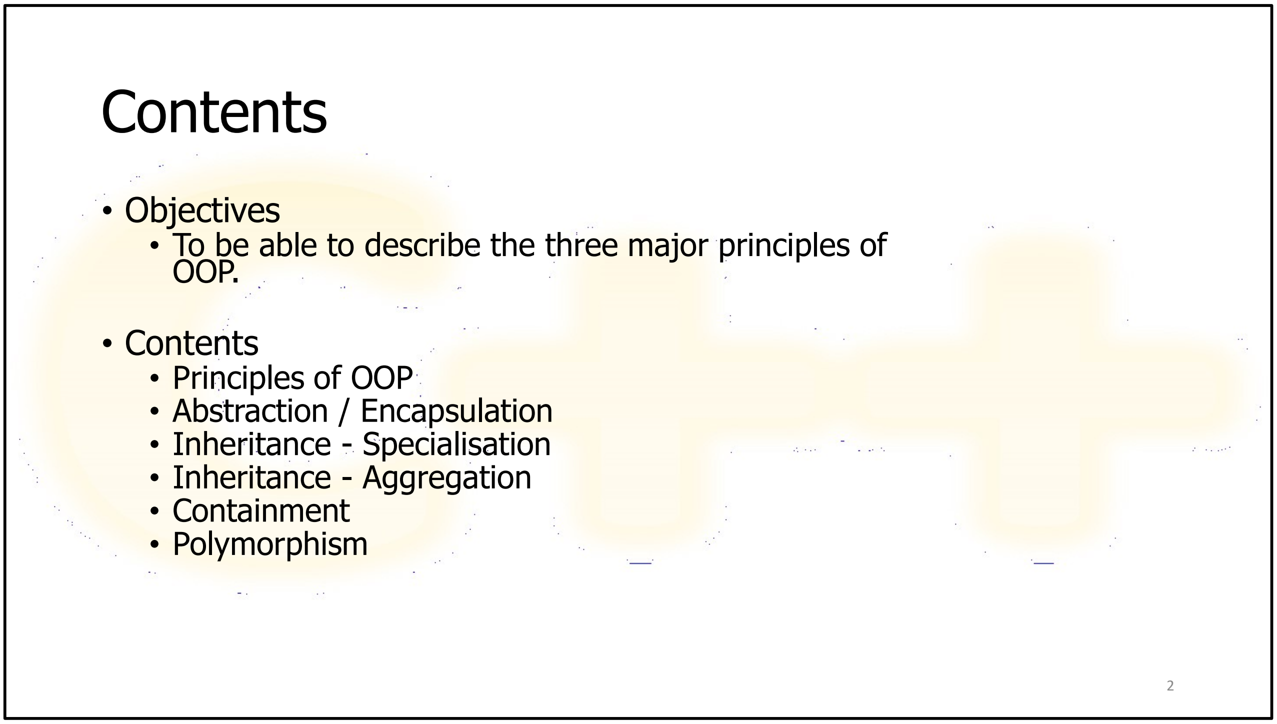
OOP

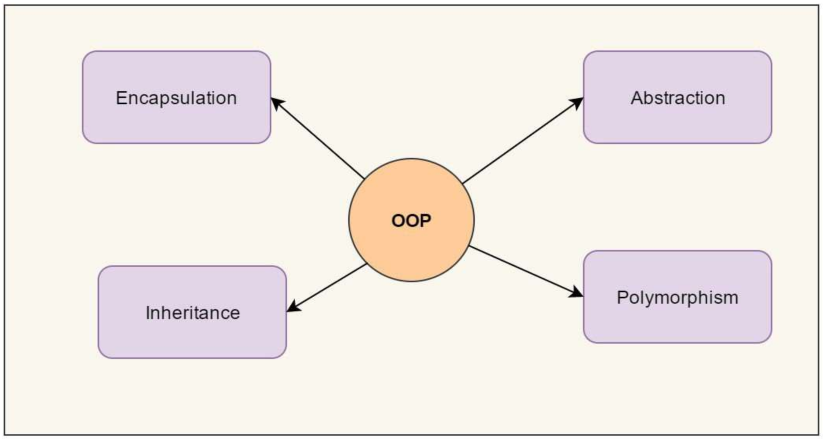


Contents

- Objectives
  - To be able to describe the three major principles of OOP.
- Contents
  - Principles of OOP
  - Abstraction / Encapsulation
  - Inheritance - Specialisation
  - Inheritance - Aggregation
  - Containment
  - Polymorphism

2

Principles of OOP



OOP

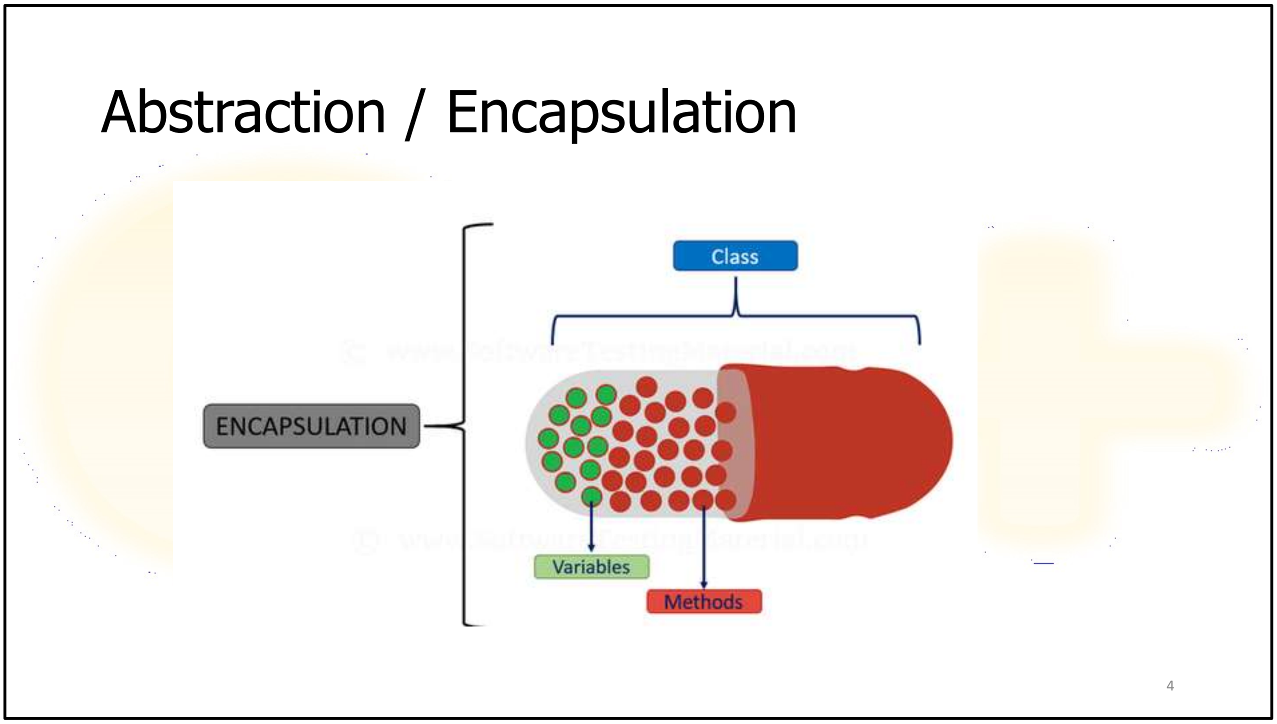
Encapsulation

Abstraction

Inheritance

Polymorphism

Four Pillars of Object Oriented Programming



Abstraction / Encapsulation

4

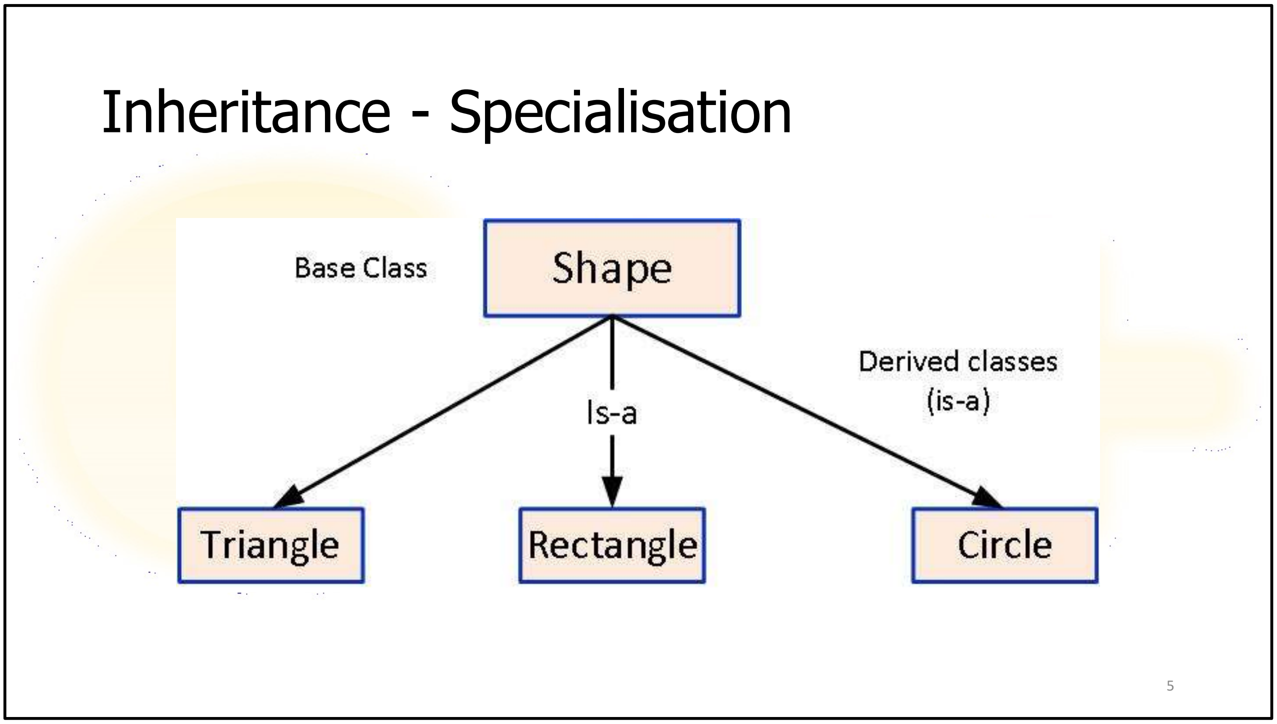
Process abstraction is the concept of being able to carry out a task without needing to know how the task is carried out in detail. For example, we all know how to turn on the TV to watch a television programme, but how does the TV work internally? The simple answer is we don't need to know. Traditional procedural languages support process abstraction through procedures and subroutines. For example consider the library of intrinsic functions usually supplied with the compiler. These functions allow the programmer to perform various complex actions without knowing the logic to perform the function. All the programmer needs to know is how to call the function. In other words process details are abstracted. Process abstraction leads to modularity. C++ lends itself to modular programming very easily as we have already seen in this course. Process abstraction can be achieved through the use of Classes that are discussed later.

Data abstraction consists of two elements. Encapsulation and Data hiding. Data abstraction is used in traditional languages such as COBOL, C etc. However it is only used for system defined data types such as Integer, Real etc.

As an example of data hiding consider a real number. There are a number of different ways that a real number can be stored in memory. It could be stored as a single or double precision number or as a packed field. The programmer doesn't need to know how a number is stored, only the name of the variable. The programmer cannot alter the way that the number is stored. In this sense how the data is structured is hidden from the programmer. This is data hiding.

Furthermore, in most high-level languages the value itself cannot be directly accessed in memory. Access is provided through the actions that are permitted by the system. E.g. assignment, addition, subtraction, multiplication and division. The real or floating point data type therefore consists of two elements: The data and how it is structured and the set of actions that the system allows to operate on the data. This is called encapsulation.

In OOP data abstraction is implemented by the use of classes.



Inheritance - Specialisation

Base Class

Shape

Is-a

Derived classes
(is-a)

Triangle

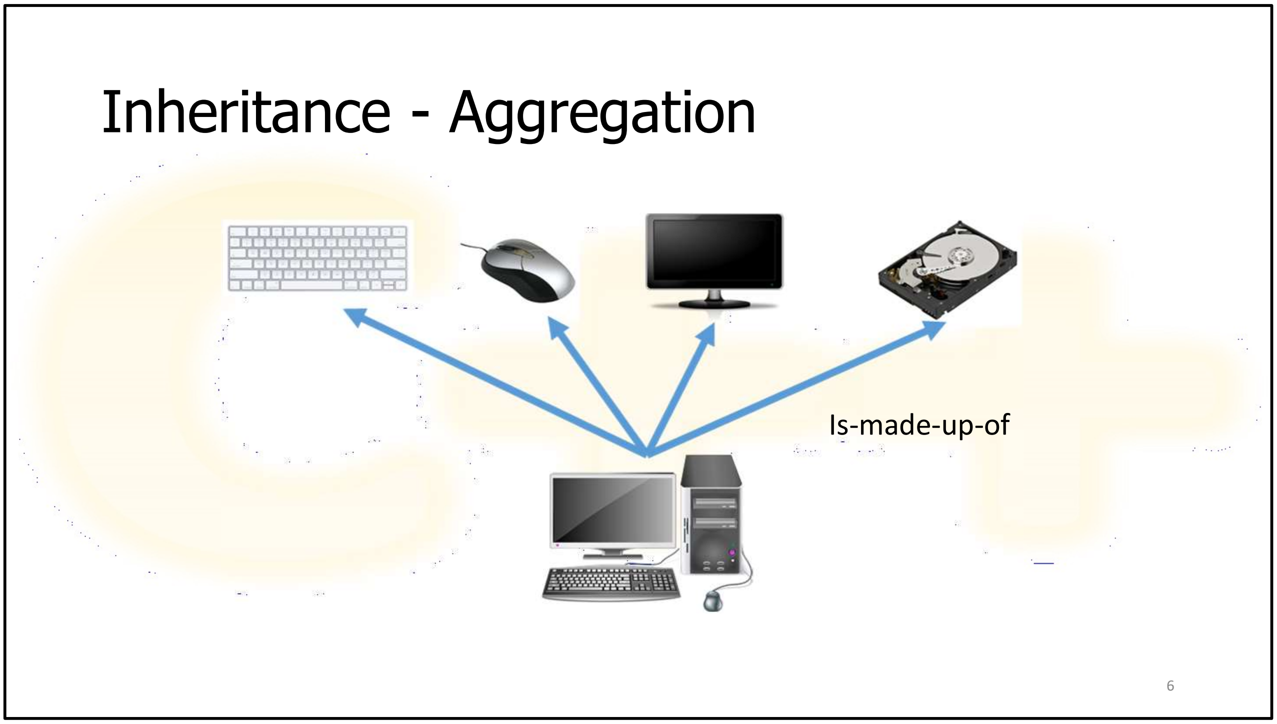
Rectangle

Circle

Classes can be organised into hierarchies. Take for instance the shape class.

The shared characteristics are located at the top of the tree and are inherited by members of the tree beneath them. Both the data and methods of a class hierarchy are inherited. All Triangles will therefore inherit the data and methods of the Shape class. Applying this principle to program design allows for code re-use and structured software. Inheritance is usually indicated in language by terms such as "is a kind of". A Rectangle is a kind of Shape.

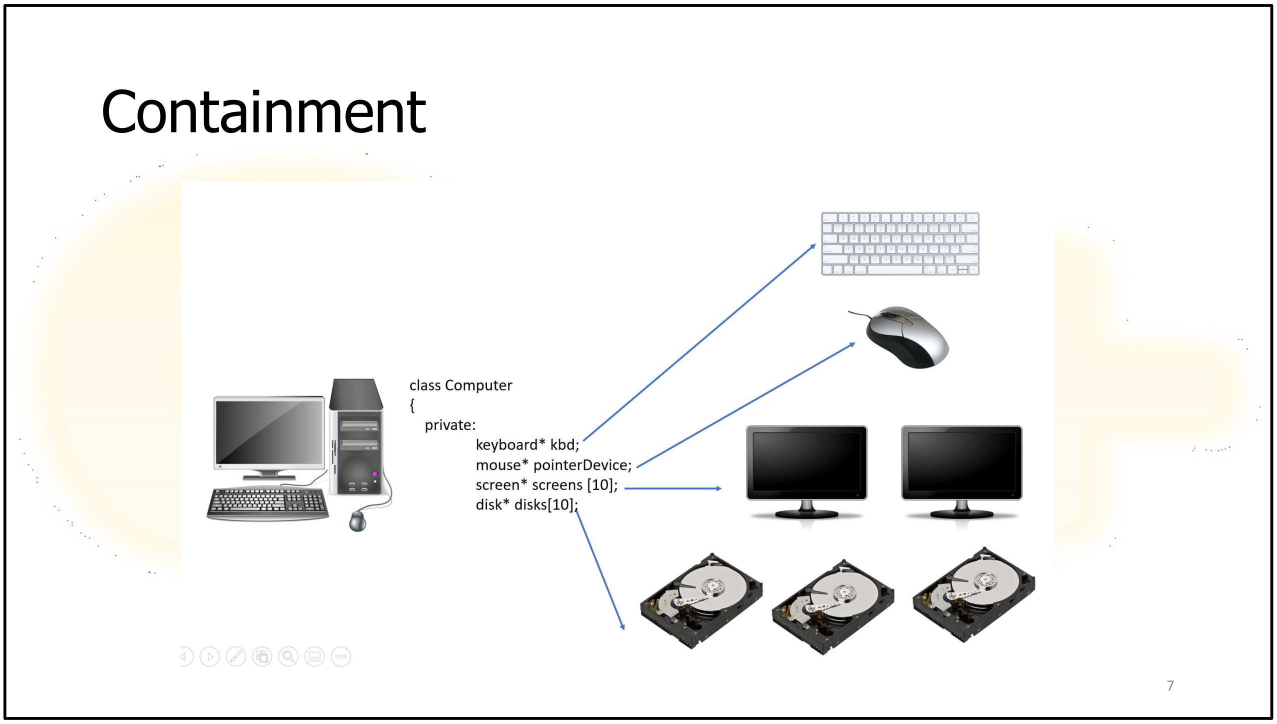
Two terms are often used in inheritance. These are base class and derived class. The Shape is the base class, and Triangle, Rectangle & Circle are derived classes.



Inheritance - Aggregation

Is-made-up-of

A computer is made up of a keyboard, mouse, screen and a disk drive. Aggregation is also sometimes called multiple inheritance. Aggregation reflects real life as complex objects are built up from smaller building blocks. It does, however, have some drawbacks. This looks fine & fits with our philosophy that complex objects are made up of smaller components. HOWEVER! What about computers with multiple screens, multiple disk drives, USB plug & play disk drives / devices. The multiple inheritance mechanism cannot cope with that type of system. With multiple inheritance you ALWAYS get one keyboard, one mouse, one display & one disk drive.

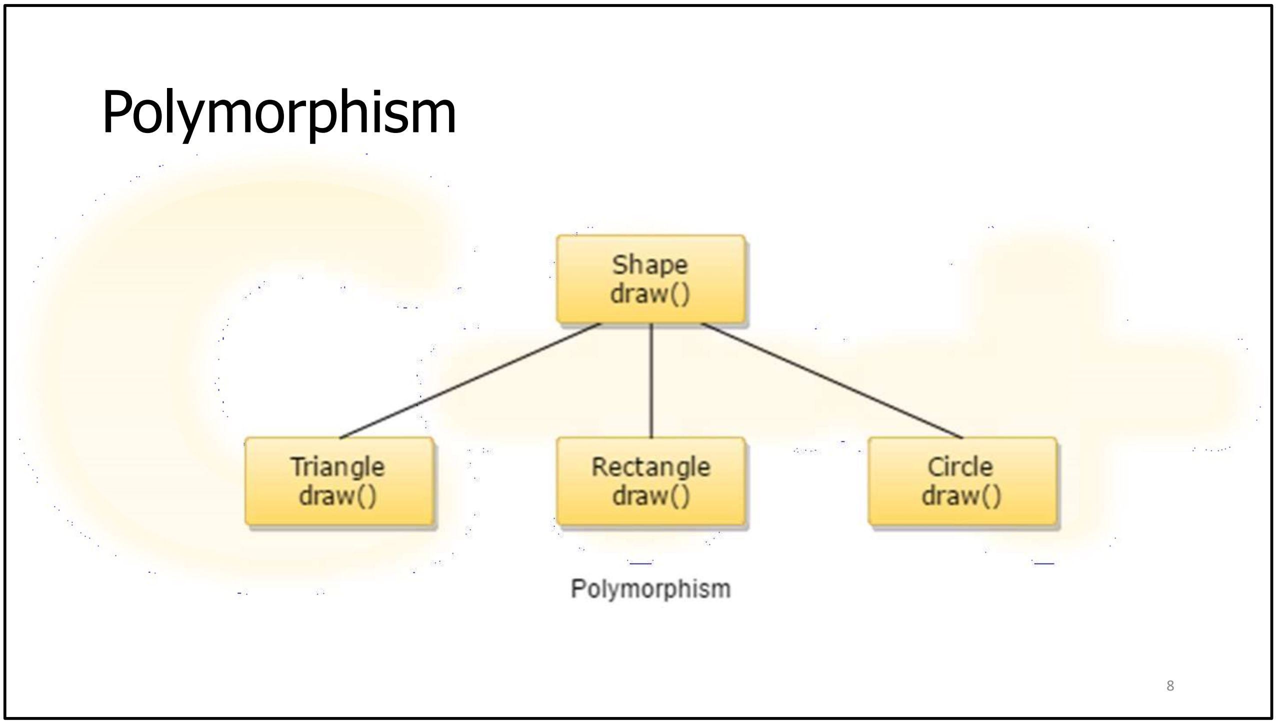


Containment

```
class Computer
{
   private:
            keyboard* kbd;
            mouse* pointerDevice;
            screen* screens [10];
            disk* disks[10];
```

This is where containment comes in. Containment should always be preferred over multiple inheritance. The Computer class contains pointers to the devices. In the case of the screen & disk we have arrays of pointers. In this case we have allowed for up to 10 screens & up to 10 disk drives.

Containment is far more versatile than aggregation. C++ is one of the few languages to support aggregation. None of the .NET languages support it.



Polymorphism

Shape
draw()

Triangle
draw()

Rectangle
draw()

Circle
draw()

Polymorphism

Polymorphism is the use of generic commands to perform actions on objects which when acted upon will carry out the commands in different ways.

Continuing with the Shape analogy, we could specify a polymorphic command draw(). If the shape is a Triangle the Triangle.draw function draws a triangle. If the shape is a Rectangle the Rectangle.Draw function is called etc.

Exercise

There is no practical exercise for this chapter.