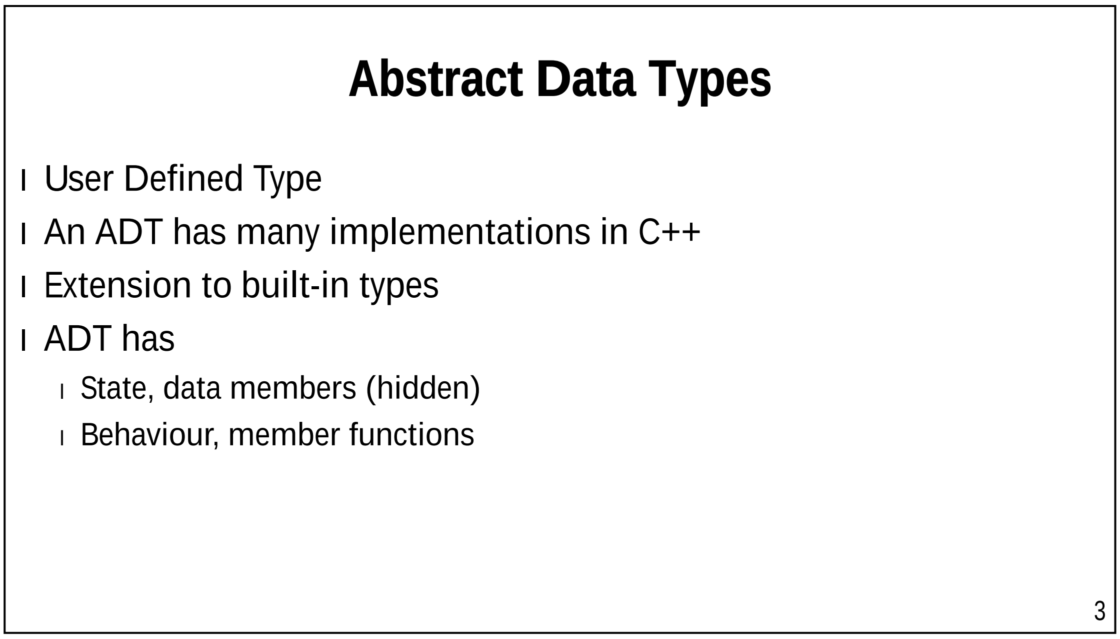
Objects and Classes

Overview

- Abstract Data Types
- Classes
- Objects
- Steps for OO Project



Abstract Data Types

- User Defined Type
- An ADT has many implementations in C++
- Extension to built-in types
- ADT has
  - State, data members (hidden)
  - Behaviour, member functions

3

Abstract Data Types (ADTs)

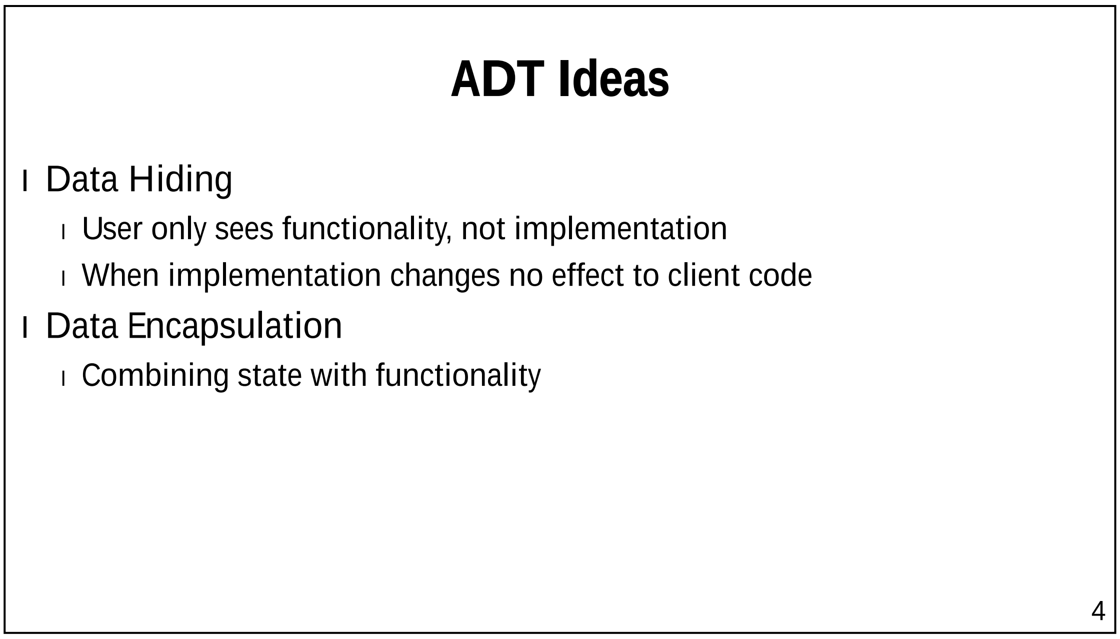
An abstract data type (or ADT for short) is an encapsulation of data and operations on that data. The study of ADTs has been used as the starting point for the development of a number of languages that support the class construct. Many object-oriented languages, such as Simula, Eiffel and C++ are based on implementation of ADTs. Other non-object oriented languages also implement ADTs. Modula 2, for example used the module to this end while Ada implements this feature using packages.
The advantages of studying ADTs are:

· They capture the essential properties of data structures.
· They can be used as specifications for the construction of classes in C++.
· They promote the use of good software engineering practices.

When designing and implementing classes in a language such as C++, beginners tend to concentrate on low level details such as how the state of objects are implemented instead to how the class interface should be created. This approach can lead to over specification. It is better to treat the problem at a higher level of abstraction and to take the view of creating an ADT that can be later implemented in C++. In this way we ensure that software quality criteria, such as correctness and robustness are carried through to implementation. An ADT is mathematically complete and correct.

Specification of an ADT

An ADT consists of two parts. The first part is a syntactic specification while the second part consists of a set of so-called axioms. The syntactic specification contains information on the names, domains and ranges associated with the type while the axioms define the meaning of the operations by stating the relationships between them.



ADT Ideas

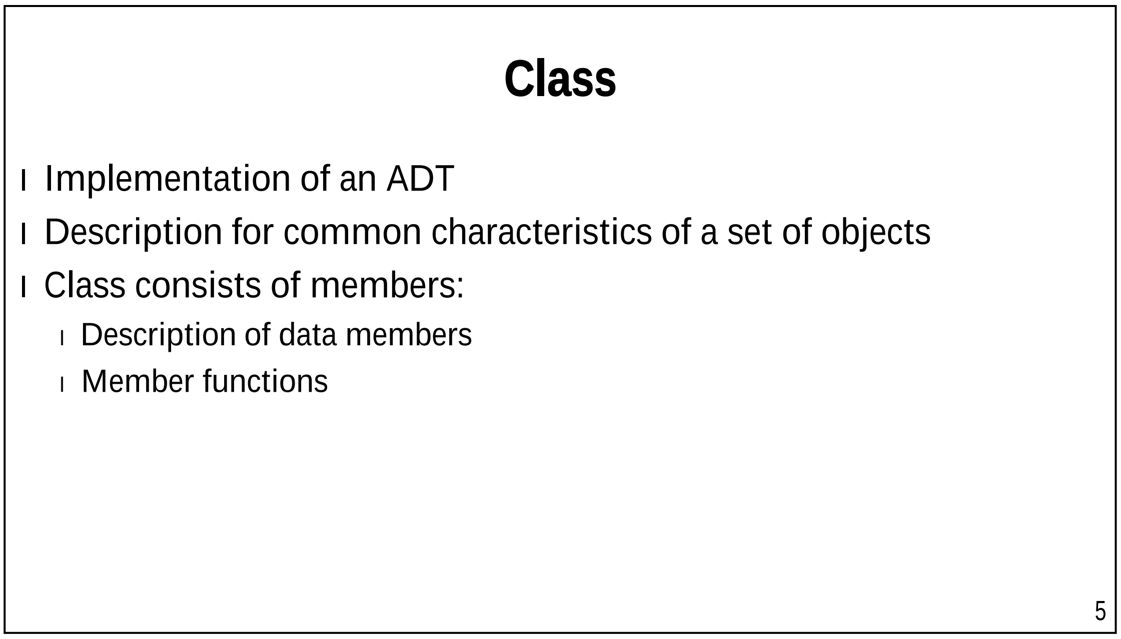
- Data Hiding
  - User only sees functionality, not implementation
  - When implementation changes no effect to client code
- Data Encapsulation
  - Combining state with functionality

4

ADT Ideas

Creating Abstract Data Types is not a new idea. It is an active subject since the 70s. The main ideas of using abstract data types are hiding something from the user. The main advantage of hiding something is that what is hidden can be changed. Sometimes people talk about black boxes when discussing ADTs and that's just what it is. The client of the ADT uses its functionality, the outside of the box and does not see how this functionality is implemented. This form is called Data Hiding, the client uses data in the abstract data type by using the outside functionality (interface) and not the data itself. When the data is changed for some reason the client does not have to change its code that uses the functionality of the data type.

Combining the data of a type and its functions is called data encapsulation.



Class

- Implementation of an ADT
- Description for common characteristics of a set of objects
- Class consists of members:
  - Description of data members
  - Member functions

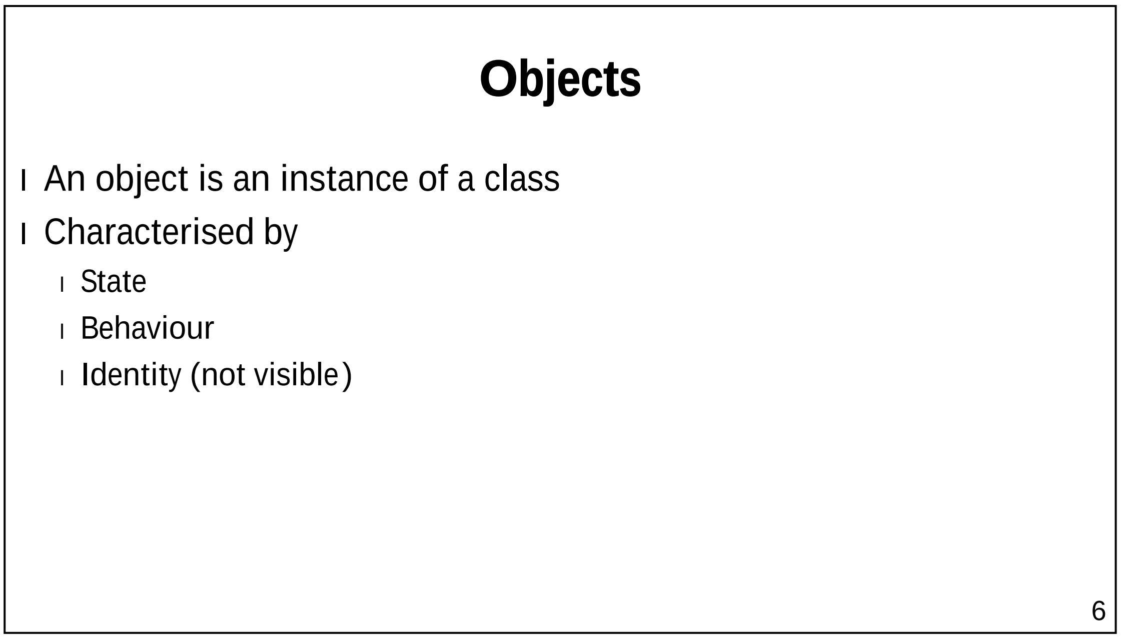
5

Class

Some class definitions:
'The structure and behaviour of similar objects is defined by their common class'

'An object models a real world entity and is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requests for services.'

'A class is an implementation that can be instantiated to create multiple objects with the same behaviour. An object is an instance of a class. Types classify objects according to a common interface; classes classify objects according to a common implementation'.



Objects

- An object is an instance of a class
- Characterised by
  - State
  - Behaviour
  - Identity (not visible)

6

OBJECTS

The central concept in the object-oriented paradigm is the object. Objects are the cornerstones of object-oriented systems and a clear understanding of their precise meaning is essential if you wish to be successful in implementing solutions in C++. The first thing to learn is how to recognise objects when you see them. The second objective is to learn how to discover, invent and find objects. This latter objective is the basis for the area of object-oriented analysis. The question of recognising objects is the topic of this section.

Objects can be characterised in a number of ways. Objects exhibit one or more of the following properties:

- They are tangible and/or visible.
- They can be comprehended.
- They can be thought about (as concepts).
- They can be acted upon.

Objects abound everywhere. We see objects in everyday life and also in areas that are both abstract and very intangible. An object can exist in someone's imagination.
Typical examples of objects are:

- Jessica
- FORTRAN
- Car
- Process number 3542
- A point with co-ordinates (1, 2)

All objects take up space. A table, for example, takes up a certain amount of space in a room. Similarly, an abstract concept in one's mind takes up space somewhere in the brain. Objects are also found in software systems (the so called internal objects in object oriented jargon) and thus consume a certain amount of computer memory.

Objects interact with one another. In this sense we see objects as small intelligent machines that can react with their surroundings. Objects react to certain stimuli and not to others. For example, points could react to questions related to geometrical properties; bank account objects would know about credit and debit information. It would however, be debatable if a point knew about bank account details. In this context we say that the behaviour of a point object consists mainly of geometrical operations.

We now treat objects in some more detail, in particular those characteristics that uniquely define them in some way. These characteristics are:

- The state of the object.
- The behavior of the object.
- The identity of the object.

In order to motivate we have decided to give an example first and then to give general definitions.

Consider a point in two-dimensional Cartesian geometry. Every such point has an x and a y co-ordinate. These two concepts belong to the state of the object (in fact, all points have a pair of co-ordinates). Each point has a specific position in the plane and this position is represented by a pair of numbers with each number representing the values of either x or the y coordinate. These also belong to the state of the point. The behaviour of the point is the set of operations that can be performed on that point. These operations would be mostly geometrical in nature, although others might be possible. The identity of a point implies that each point is uniquely distinguishable from every other point. This is a difficult concept to grasp on a first encounter but it basically means that we can always refer to a point once it has been created and before it has been destroyed. Note that it is possible for two distinct points to have the same coordinates! This is because the coordinates of a point belong to its state and has in principle nothing to do with the identity of that point.
Having constructed a simple example to exemplify the characteristics of a point, we state the general results.

The state of an object includes all of its properties plus the current values of these properties. Put roughly, the state of an object corresponds to its structure. In a number of books, the term attribute is sometimes used as a synonym for state.

The behaviour of an object is determined by how it reacts to messages from other sources.

The identity of an object is that property which distinguishes it from all other objects. Object oriented languages relieve the programmer of having to set up an administration so that each object in a given software system has a unique identifier. This is implemented differently in non object oriented systems and is achieved in a number of ways, for example:

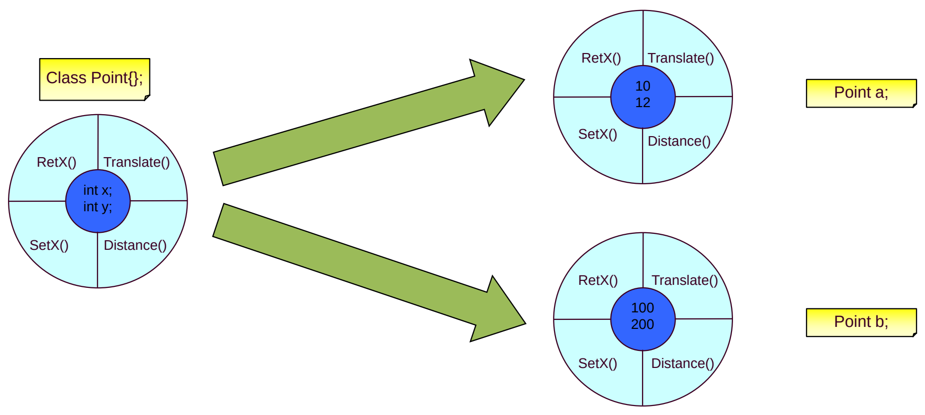
- Memory references and addresses.
- User specified names.
- Identifier keys in databases.

To sum up, we can give a 'formula' for an object as implying the following ('= =' means equality):

object = = state + behaviour + identity

We have mentioned that objects are concrete and tangible; there are many objects that are similar to each other in some way. For example, all the points in two-dimensional space have similar state and behaviour and could in this sense be amalgamated into some larger, more general abstraction. This is what is done in practice and the resulting edifice is called a class. We can give a name to a class and in this specific case we decide to call it POINT. Thus the class POINT consists of all points in two-dimensional space. Thinking about POINT is easier than having to think about all points at once (there is an infinite number of them!).

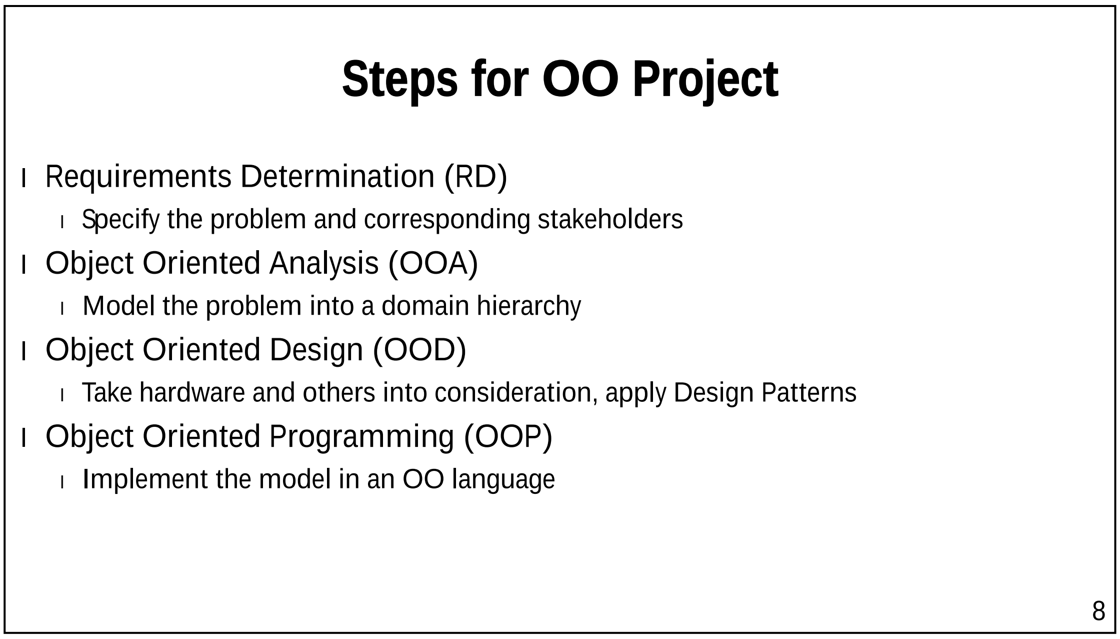
Objects and Classes



7

Representation of an Object

In this course we will use the shown picture for representing objects and classes. The hole in the middle is the state (data members) and the outer part of this donut represents the behaviour (member functions).



Steps for OO Project

- Requirements Determination (RD)
  - Specify the problem and corresponding stakeholders
- Object Oriented Analysis (OOA)
  - Model the problem into a domain hierarchy
- Object Oriented Design (OOD)
  - Take hardware and others into consideration, apply Design Patterns
- Object Oriented Programming (OOP)
  - Implement the model in an OO language

8