Script for the vhb lecture

Programming in C++
Part 1

Prof. Dr. Herbert Fischer
Deggendorf Institute of Technology

Table of Contents

5 Paradigms of Object Orientation (OO) 1

5.1 Overview 1
5.1.1 How did OOP originate?? 1
5.1.2 What is OOP used for? 1

5.2 The main basics 2
5.2.1 Object-oriented thinking 2
5.2.2 Data abstraction 3
5.2.3 Encapsulation 4
5.2.4 Inheritance 5

5.3 Advantages of the object-oriented approach 6

5.4 Objects 6
5.4.1 The term "object" 6
5.4.2 Data member types 7
5.4.3 Member function types 7
5.4.4 Object diagrams 8

5.5 Classes 8
5.5.1 The term "classes" 8
5.5.2 Class diagrams 9

5.6 Inheritance 10
5.6.1 The principle of inheritance 10
5.6.2 Inheritance hierarchy 11

5.7 Concluding Example 12

5 Paradigms of Object Orientation (OO)

Chapter 5 teaches you the characteristics and basic concepts of object orientation.

5.1 Overview

5.1.1 How did OOP originate??

Initially computers were exclusively used to solve numerical assignments. Their task was to solve arithmetic problems whose scope and complexity reached or even exceeded the limits of human arithmetic skills. A typical example of this use is the calculation of log tables, a task many of the first computers were designed for. Log tables were needed for many calculations in the military and engineering sciences, and the hand-written computing tables were mostly incorrect. For these numerical purposes, a programming concept, which on the one hand provided a set of data (given and calculated ones) and on the other hand provided functions to change them, was quite sufficient. In addition, the programs were still relatively small and designed for a specific utilisation time, the so-called software lifecycle.

With demands on more powerful computers increasing, however, the scope of programs grew considerably. In addition, the applications were no longer limited to solving numerical problems. The computer also transformed itself into a universal work tool, up to multi-media, artificial intelligence and computer-mediated communication.

This development also required rethinking the programming concept. The complexity of the software did not only require a better overview of programming.
It was also no longer practicable to re-program software from scratch every time, only to have a product that was very rigid.
The program could only be replaced if it was outdated after a few years.
It became necessary to divide software into small units of meaning that can be used over and over again. A division into data set and functionality, as it was the common practice in the past, turned out to be very unwieldy. Instead, the program parts were structured into functional units containing the data required for them. The concept of OOP was born.

5.1.2 What is OOP used for?

In the previous paragraphs many of the motives to use OOP were already mentioned. This section is intended to summarize them, as OOP newcomers in particular are often not familiar with the concepts of OOP. Usually, the purpose of this programming concept is not clear to them.
The basic idea of OOP is to create a better structured program code by creating units of meaning (objects). These units of meaning consist of data (data members) and functions (member functions). This creates an easily definable interface to the objects, which simplifies reuse. Subsequent changes can also be made easily as long as they do not change the existing interface (an extension is of course possible).

In addition, the inheritance concept (derived classes inherit all properties of the base class) enables you to easily extend existing objects with new functions or create variations of existing objects.

OOP tries to adapt to the human way of thinking. The basic idea of OOP is as follows:
- The world consists of acting objects that can be grouped into classes. These objects are standardised for programming purposes. The classes are created during standardisation. An object is always an instance of a class, so an object is created from a class.

- Example: A person's data: *Name: Willi, Age: 20* is stored in an object. This is an object of the person type person, which is the class.

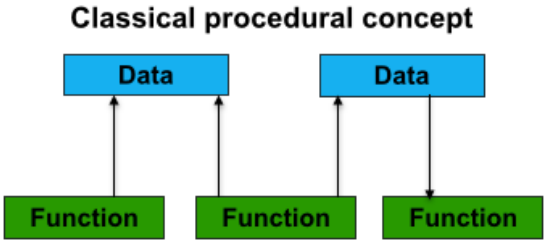
Objects also have properties and can communicate with each other. In order to enable an OOP programmer to achieve an easier and more daily-life oriented abstraction in the implementation, an attempt is being made to perceive reality to a certain extent.

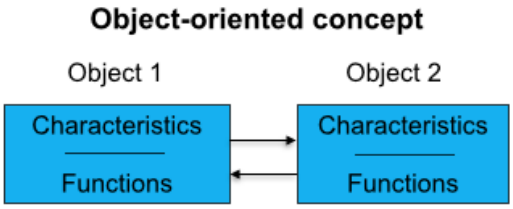
However, OOP also has disadvantages:
- Modelling class hierarchies or inheritance diagrams takes a lot of time because, in contrast to procedural languages, planning has a high priority in OOP.
- On the other hand, OOP applications are mostly also more computational and memory intensive and therefore often run more slowly than, for example, C applications.

Due to the ever progressing development and in order not to miss the connection to an increasingly growing object-oriented software industry, one should be proficient in OOP languages such as Java or C++.

5.2 The main basics

What exactly is OOP then?





5.2.1 Object-oriented thinking

Traditional procedural programming is characterised by the fact that data and functions that process this data do not form a unit. This results in a greater susceptibility to errors, e.g. due to incorrect access to data or the use of data that has not been initialized. In addition, there is also a high maintenance requirement, for example when the data has to be adapted to new requirements.

In contrast, in OOP the objects form a unit of data (properties) and functions (functions, abilities).

This results in decisive advantages for the software quality:
- Higher reliability
- Less maintenance expenditure
- Improved reusability

Characteristics of OOP:
An object-oriented programming language is characterized by having language elements to support the following OOP paradigms:

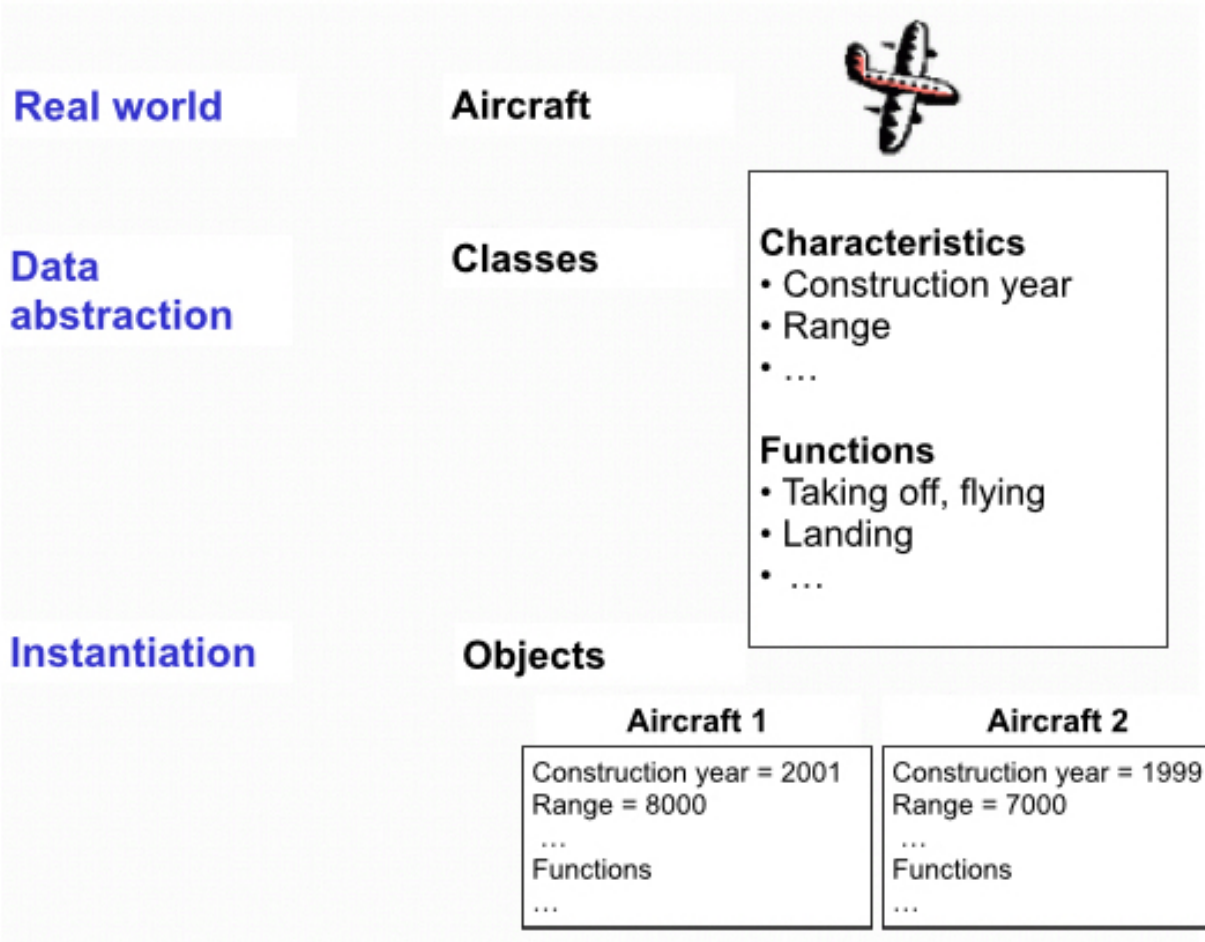
- **Data abstraction**
  Abstract data types (classes) can be defined, which describe the properties and abilities of objects.

- **Encapsulation**
  Members of an object can be protected against uncontrolled access from the outside world. For communication with the outside world, i.e. with other objects, each object has a »public interface«.

- **Inheritance**
  New objects can be derived from existing objects. They »inherit« the existing characteristics and abilities that can be extended and changed.

5.2.2 Data abstraction

One way of dealing with complex situations is abstraction. Properties and processes are reduced to the essentials and provided with generic terms.



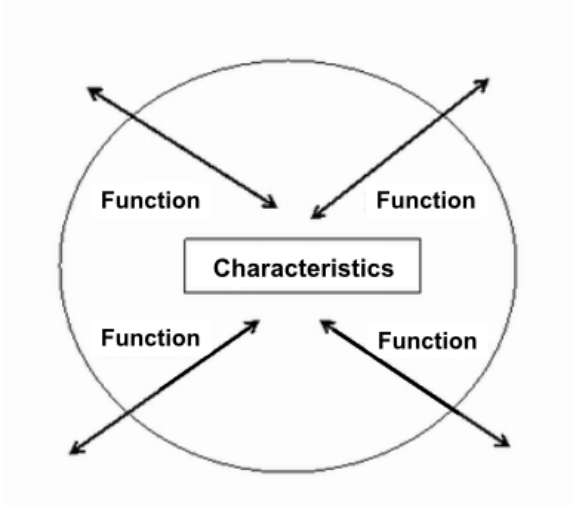
An aircraft is an example of an abstraction carried out in various ways:

- An aircraft is a collection of different parts such as engine, wings, wheels, etc. In addition, each aircraft has characteristics such as range, maximum speed and transport capacity. The current status, such as »The aircraft is parked«, is also part of the characteristics.
- Of course, the capabilities of an aircraft primarily consist of taking off, flying and landing. In addition, an aircraft must be refuelled, loaded, unloaded and maintained.
- An aircraft is also the generic term for different types of aircraft such as gliders, aeroplanes, jet fighters, transport plane, passenger aircraft, etc.

If the details of an aircraft or the differences between different types of aircraft are not of interest, simply the abstract term »aircraft« is used. This means that the typical activities of an aircraft can be brought to the fore: An aircraft can be built, maintained or dismantled. An aircraft takes off, flies, e.g. from Munich to Cologne, or lands.

5.2.3 Encapsulation

As already described in the introduction, one of the central aspects of OOP is to merge data with the associated functions. This process is called ***encapsulation***; the resulting data structure is called ***class***. A variable or constant that uses this data structure is called an ***instance*** of the class. In addition to the data, a class can also perform comprehensive functionalities. In this way, it becomes an ***abstract data type*** that does not only specify the data itself, but also the possible ways of dealing with it (functions). The data of a class is called *data members*, and the functions are called *class member functions*.

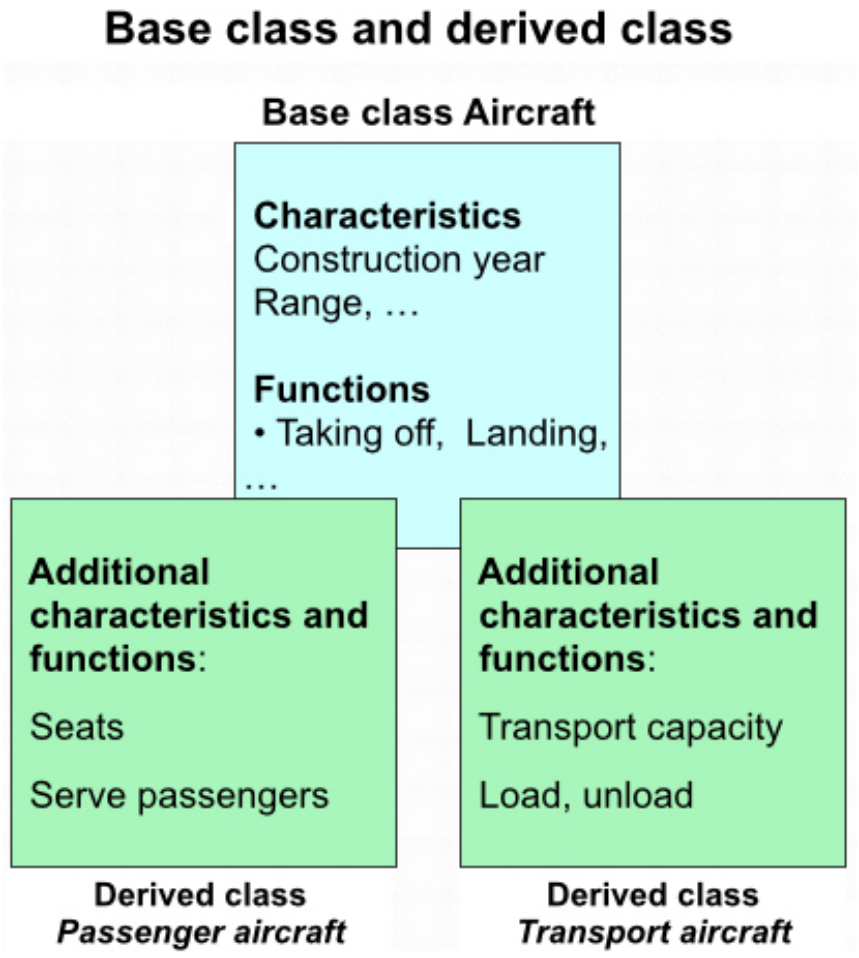


Objects have an inside and an outside. Inside means that there are data members and perhaps member functions that the user (i.e. the calling program part) of this object does not see, whereas the outside is the set of data members and class member functions that the user can see. Object-oriented languages usually have the chance to make such a distinction. In C++ data members and class member functions that can only be accessed from the inside of a class are labeled as ***private***, members that can be accessed from anywhere outside the class as ***public***. The use of private data members is only possible by using public functions. In a consistent object-oriented programming style, all data members should be declared as private and access to the data members should only be possible using class member functions (so-called *access functions*). This ensures that you can later exchange the data members of the objects without changing the calls of the corresponding class member functions. Therefore you do not have to change any program that uses this object. It is also possible to include additional features into the class member functions, such as a flag that indicates whether the data has been changed. If direct access to the data members of the class was possible, this flag would be unreliable.

The concept of the inside and outside of an object indicates that an object has an exterior. This exterior is formed by the access options to functions and data members of the object. They are primarily determined by the names of the public data members and member functions, as well as by the parameters of these functions. The set of access options is referred to as an ***interface***.
Calling a member function of an object is often referred to as a ***message***. This is based on the idea that a program part causes an object to do something by sending a message to it. For example, this can simply be the adding of a data member. In this programming concept it is, however, essential that the object will never be changed by an external part of the program, but that the part only sends a message to the object telling it to do so itself. This means that an object can always retain control of its inside.

5.2.4 Inheritance

Another important concept in object-oriented programming is the option of ***inheritance***. To avoid having to program a completely new class every time, again and again, in object-oriented languages you can specify that a class is to 'inherit' all data members and member functions of another class, i.e. the new class contains all the data and functions of the old class from the outset. These can then be extended by further data members or member functions or, if necessary, overwritten. This creates a relationship such as 'an apple is a fruit', and additional data members and member functions are used to specify the new class more precisely. The old class is referred to as ***base class***, the new class is referred to as ***derived class***, and the inheritance structure is referred to as ***class hierarchy***.



5.3 Advantages of the object-oriented approach

Since the early 1990s, the importance of object-oriented analysis and design methods has steadily increased and is nowadays the basis for most software developments - especially when it comes to large or complex systems. In object-oriented software development, the results of the analysis, design and implementation phases are created in an object-oriented way. A big advantage of object-oriented concepts is that they can be used in all phases of the development process. The implementation also takes place in an object-oriented programming language such as JAVA or C++. In addition, the user interface of the software can also be object-oriented, and object-oriented databases can be used. This results in software development "as an integrated whole".

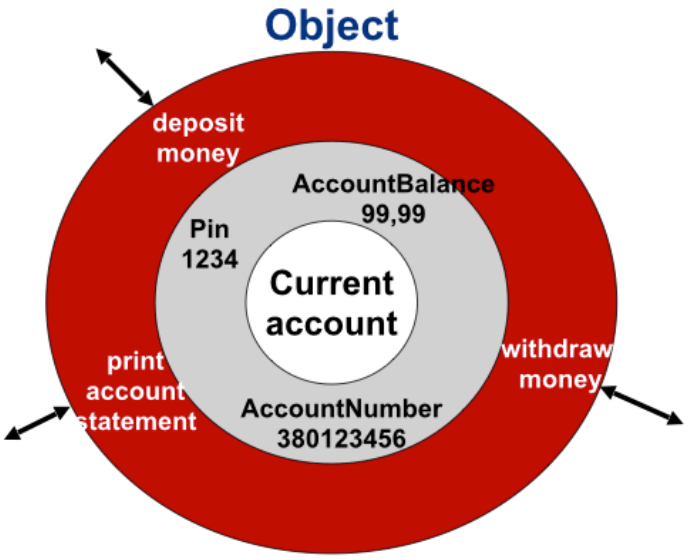
In the end, software development always means abstraction, too. A concrete question from the real world must be transformed into an abstract model of the computer, which is implemented at the lowest level with bits and bytes of the computer. It is much easier to create an abstract model of the real question by identifying the objects of the real world and implementing them directly, using object-oriented programming languages.
If an object-oriented analysis is already carried out and described in an object-oriented analysis model, it is relatively easy to obtain an object-oriented design. And, if this design has been carefully developed, the necessary implementation in C++ or JAVA can be obtained very easily – in parts even automatically.

The better consistency of object-oriented techniques is achieved by using the same concepts in all phases of the development process. In the end, the biggest advantage of object-oriented software development is the fact that software requirements can be implemented more easily, i.e. with less effort and fewer errors.

5.4 Objects

5.4.1 The term “object”

Let’s start with a review of the basic term "**object**".
Each object has a unique **name** to identify it. An object contains values that are stored internally; these values are referred to as **data members** of the object. The processes are called dynamic properties, **behaviour or member functions** of the object.



In our example, the objects **name** is "Current account”. The **data members** of the object are:

- “AccountBalance=99,99”
- “AccountNumber=380123456”
- “Pin=1234”

The **member functions** are “deposit money”, “withdraw money” and “print account statement”.
The data members of “Current account” can only be accessed and modified by using its “own” member functions.

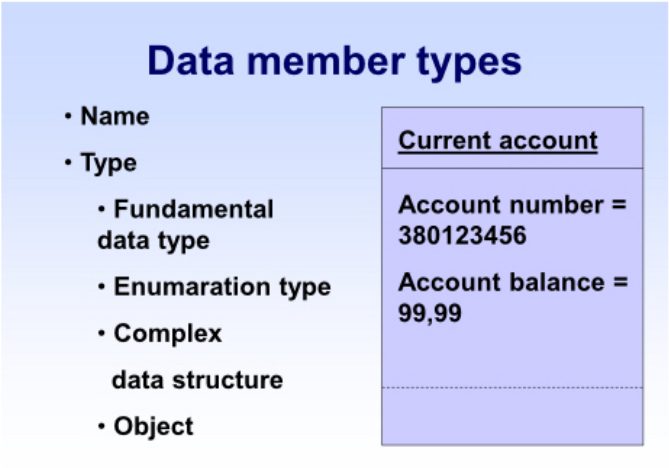
This means:
- An object encapsulates data members and member functions.

One can also say:
- An object encapsulates its own data and operators.

This basic principle is also known as **limited access**.
An object can communicate with other objects using the member functions and their operators.

5.4.2 Data member types

Let's take a closer look at the data members of an object. **Data members** describe the **data** of an object and are defined by their **names** and **types**.



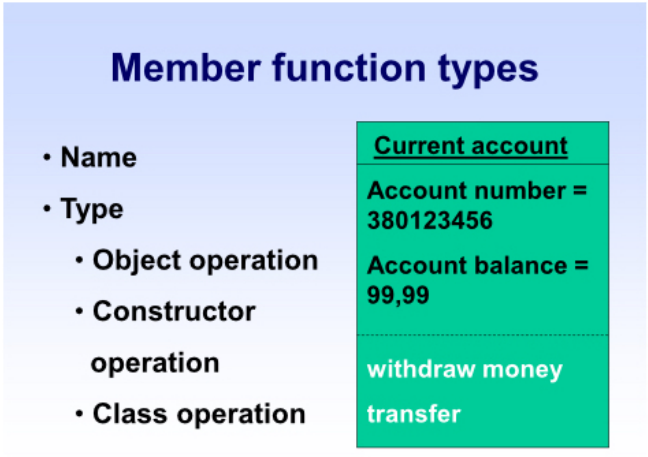
In C++, **data members** can be declared as any fundamental data type such as "integer", "float" or "char". In our example, the account number of type "integer" and the account balance of type "float" could be defined this way.

However, data members can also be defined as enumeration types, complex data structures or even as objects.

5.4.3 Member function types

Member functions are functions that are executed by an object. They are also known as operations.

Each operation can access all data members of an object. Member functions are always bound to objects! Therefore, a member function can only be called in conjunction with an object.



There are two types of operations:

- **Object operations**

They are always used for an existing object. In our example, the operations "withdraw money" and "transfer money" are defined.

- **Constructor operations**

They create a new object and perform initializations and data insertions. For example, the operation "open cash account" is a constructor operation.

In the course of the basic concept "classes" we will also get to know the
- **class operations**

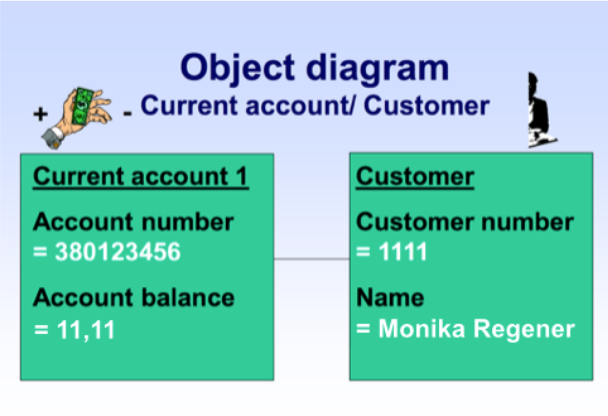
Is it possible to add new data members and member functions to an object at a later time?

Yes, that is exactly the advantage of OOP!

The limited access principle ensures that new data members and/or data member values can be added to an object without informing the "outside world". For example, you can add the data member "PIN for online banking" to the object “Current account”. This process remains hidden from other objects. The implementation of new member functions is also possible at any time.

5.4.4 Object diagrams

Object diagrams represent objects and their relationship to each other.



In our example, we illustrate the relationship between the object "Current account" and the object "Customer" in an object diagram.

The object diagram shows the objects, the corresponding data members and their values, and the relationship between the objects.

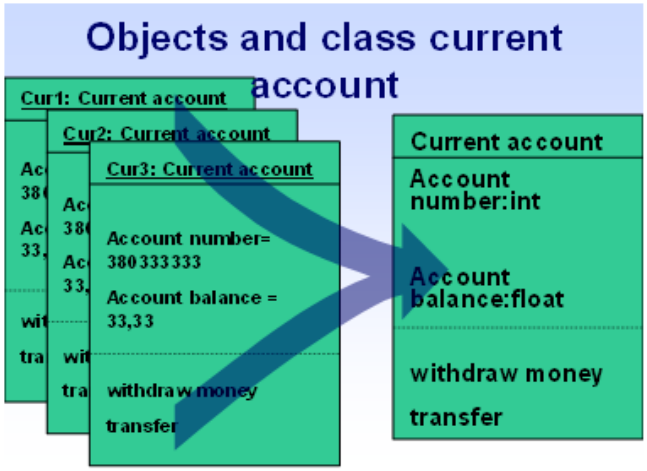
The specification of the member functions is optional.

5.5 Classes

We will now explain another basic concept, which is based on the concept of objects, the **classes.**

5.5.1 The term “classes”

A **class** defines corresponding data members, member functions, and relationships for a collection of similar objects. The class name should be unique throughout the entire program code. A class has the ability to create new objects. Each object belongs to exactly one specific class.



Let’s take a look at an example from the "world of bank accounts".

Current accounts 1,2 and 3 can be grouped together to one single class "Current account". The member functions "withdraw money" and "transfer money" are valid and accessible to all the objects of class "current accounts". The class data members, such as account number and account balance, apply to all objects of the class. The values of the data members of each object of this class are different.

What types of member functions are available for operating within a class or between different classes?

In the following, one example each will be used to describe the different types of member functions:

- Operations with read access to data members of the same class
  Example: print account statement

- Operations with write access to data members of the same class
  Example: deposit money

- Operations to perform calculations
  Example: determine average account balance of a specific current account

- Operations to create and delete objects
  Example: open new current account

- Operations that select objects of a class according to particular criteria
  Example: sort checking accounts by account balance

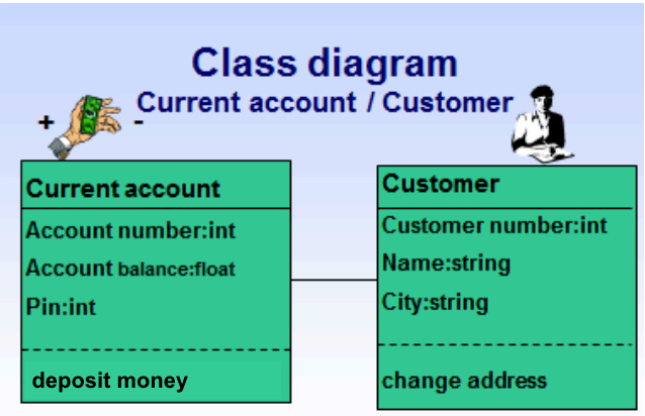
- Operations to activate the operations of other classes
  Example: create withholding tax certificate

- Operations to connect objects with each other
  Example: connect to online access

The **name of a class operation** must be unique within the class. Outside the class, the operation is called “CLASS.OPERATION”.

5.5.2 Class diagrams

Class diagrams represent classes and their relationship to each other.



In our example, we illustrate the relationship between the class "Current account" and the class "Customer" in a class diagram.

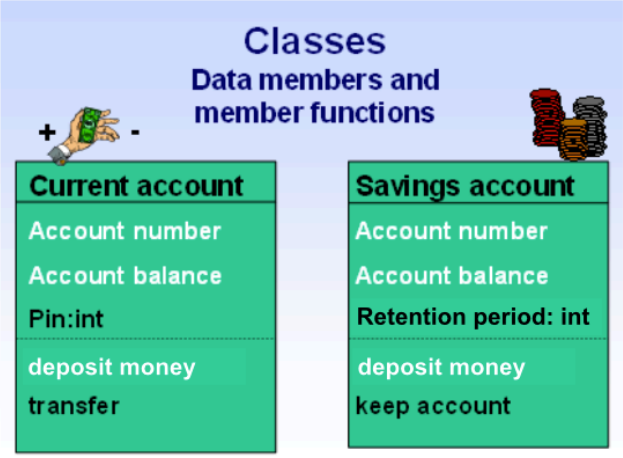
The classes are represented in the diagram with their data members and class operations (class member functions). In contrast to the object diagram, the data members are specified with their data type, but without any values. The initialization with specific values only takes place when objects of a class are created.

5.6 Inheritance

We will now discuss another basic concept, which results from the abstraction of classes, the **principle of inheritance**.

5.6.1 The principle of inheritance

Inheritance describes a relationship between a general class (also known as the base class) and a specialized class.
Let's first have a look at the principle of inheritance on the basis of a practical example.



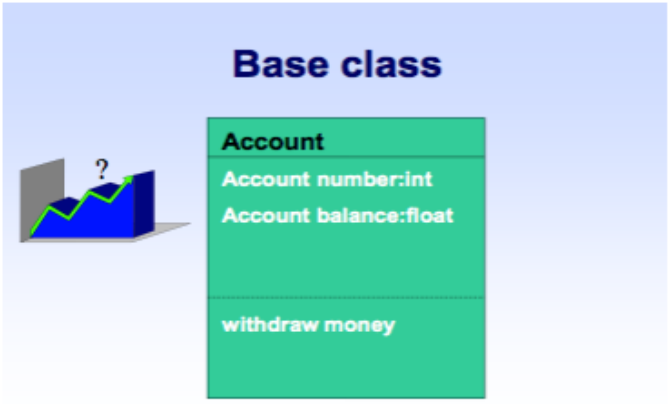
Current accounts and savings accounts have common, but also different characteristics (data members) and class member functions.

Both account classes can be identified their an account numbers and show the current balances. Only for the current account you will receive a PIN and only the savings account may have a blocking period.

You can deposit money in cash or non-cash on both account classes, but a savings book is only kept for the savings account.

There seems to be a base class describing the common data members and member functions, and specialized classes with additional data members and member functions.

What could the base class look like in our example?



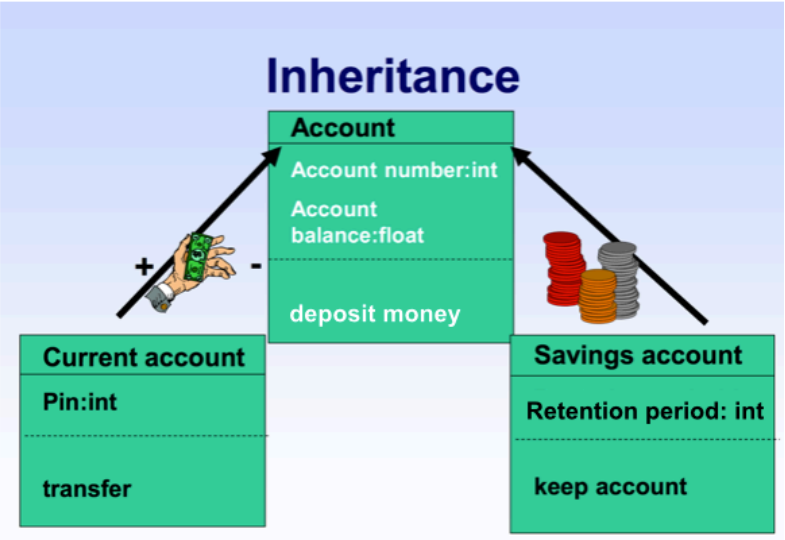
Right !

The base class called "Account" has the data members "Account number" and "Account balance" and the class member function "deposit money".

The base class inherits all its data members and class member functions to the derived classes, which in turn are extended by their own local data members and member functions.

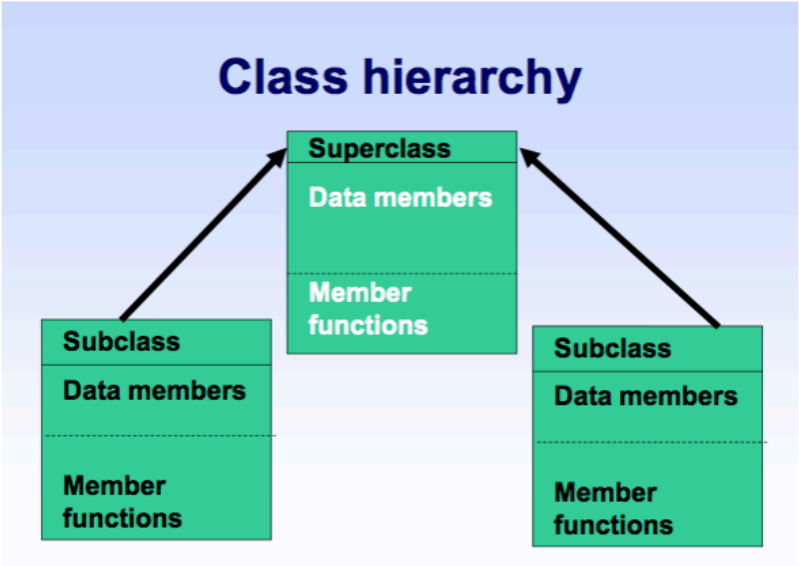
5.6.2 Inheritance hierarchy

The representation of inheritance is also referred to as a **class hierarchy** or **inheritance structure**.

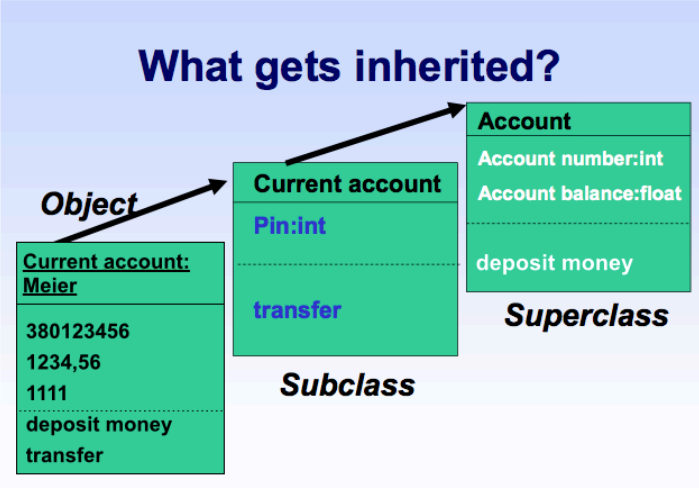


In summary, you could say:
The general class (base class) is also referred to as **superclass**, whereas the specialized class (derived class) is referred to as **subclass**. Combing shared characteristics from subclasses into a generalized superclass is called **generalization,** whereas creating new subclasses from an existing superclass is called **specialization.** A class hierarchy can of course be built up using several hierarchy levels. Each object of a subclass **is also an** object of the superclass.

Note:
The arrows in the class hierarchy always point towards the superclass.



To explain what data members and member functions are inherited to whom, these bank accounts serve as a suitable example.



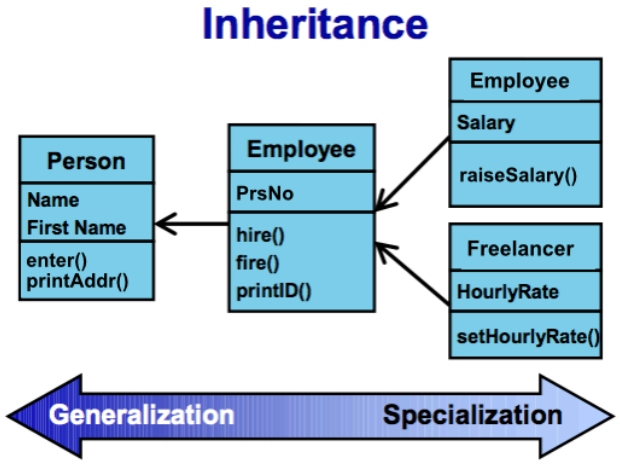
The data members and class member functions of the subclass “current account” and the superclass “account” are passed on to a concrete object of the class “current account” (a concrete current account containing concrete values).

5.7 Concluding Example

In a project management system, which is to distinguish between employees and freelancers, the concept of inheritance can be used advantageously. A relationship between a general class, such as our staff member, and a specialized class, such as the employee or the freelancer, is called inheritance. Using the principle of inheritance, we can build complete class hierarchies, starting from a general base class up to more specialized classes.

Example:

- Person
- StaffMember
- Employee Freelancer
- PaidAfterPayScale PaidWithoutPayScale



The UML represents the connection between superclasses and subclasses by means of an arrow pointing towards generalization. Whenever a relationship between classes can be described as "is a", there is a generalization.
The "is a" also means that a more specialized subclass can always be used like its base class. In our example of project management this means that a freelance employee can also work on a project, since he or she is also an employee that is connection to the project.

Now we have come to know the most important basic concepts of object-oriented software design.