Introduction to C++ and Object Oriented Programming

Wouter Verkerke (NIKHEF)

v55 – Edition for 2015 Master course

0 Introduction & Overview

Intended audience and scope of course

- This course is targeted to students with some programming experience in procedural (i.e. non-OO) programming languages like Fortran, C, Pascal
  - No specific knowledge of C, C++ is assumed

- This course will cover
  - Basic C/C++ syntax, language features
  - Basics of object oriented programming

- This course has some extra focus on the application of C ++ in (High Energy) Physics
  - Organized processing and analysis of data
  - Focus mostly in exercises

Programming, design and complexity

- The goal of software – to solve a particular problem
  - E.g. computation of numeric problems, maintaining an organized database of information, finding the Higgs etc..
- Growing computational power in the last decades has allowed us to tackle more and more complex problems
- As a consequence software has also grown more powerful and complex
  - For example Microsoft Windows OS, last generation video games, often well over 1.000.000 lines of source code
  - Growth also occurs in physics: e.g. collection of software packages for reconstruction/analysis of the BaBar experiment is ~6.4M lines of C++
- How do we deal with such increasing complexity?

Programming philosophies

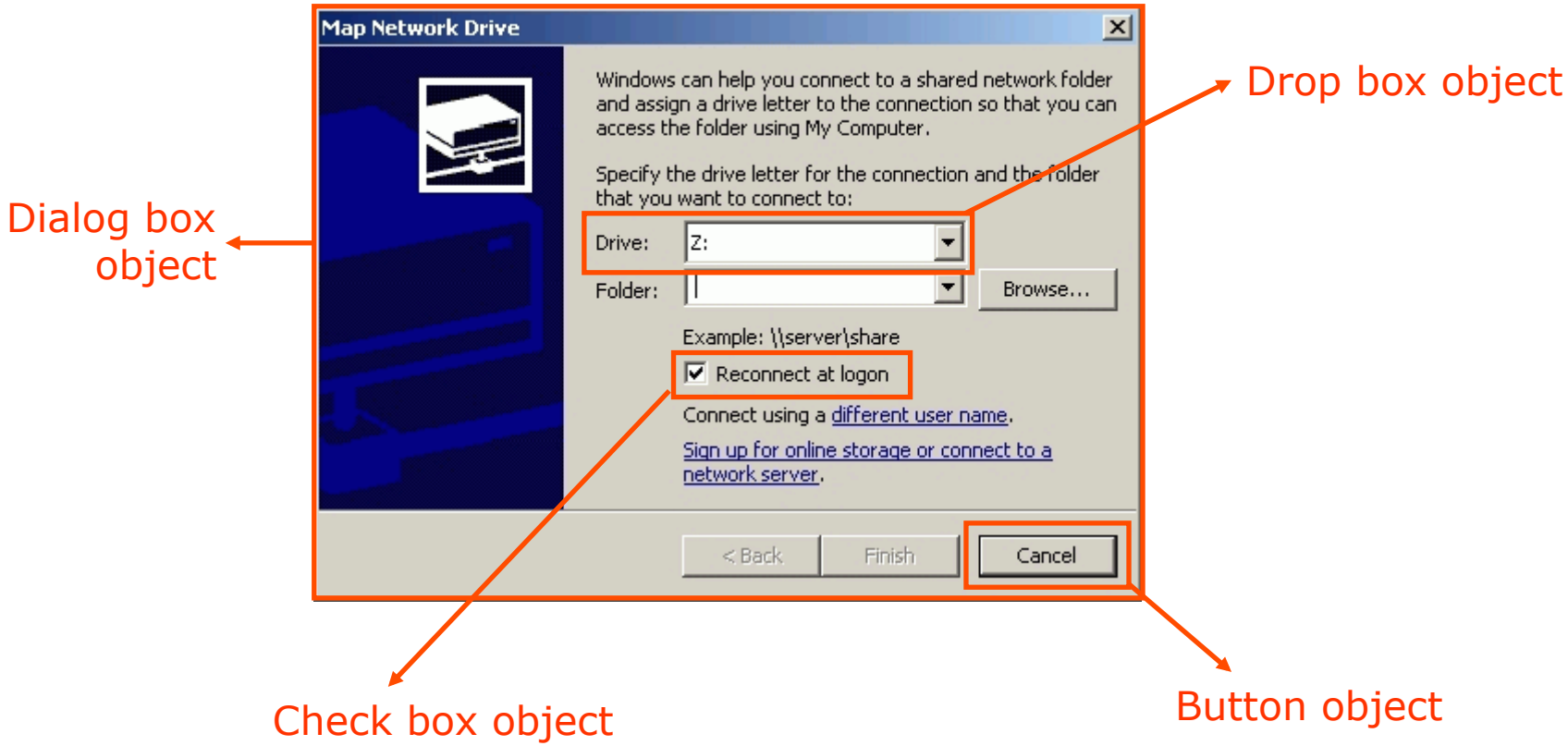
- Key to successfully coding complex systems is break down code into smaller modules and minimize the dependencies between these modules

- Traditional programming languages (C, Fortran, Pascal) achieve this through procedure orientation
  - Modularity and structure of software revolves around 'functions' encapsulate (sub) algorithms
  - Functions are a major tool in software structuring but leave a few major design headaches

- Object-oriented languages (C++, Java,…) take this several steps further
  - Grouping data and associated functions into objects
  - Profound implications for modularity and dependency reduction

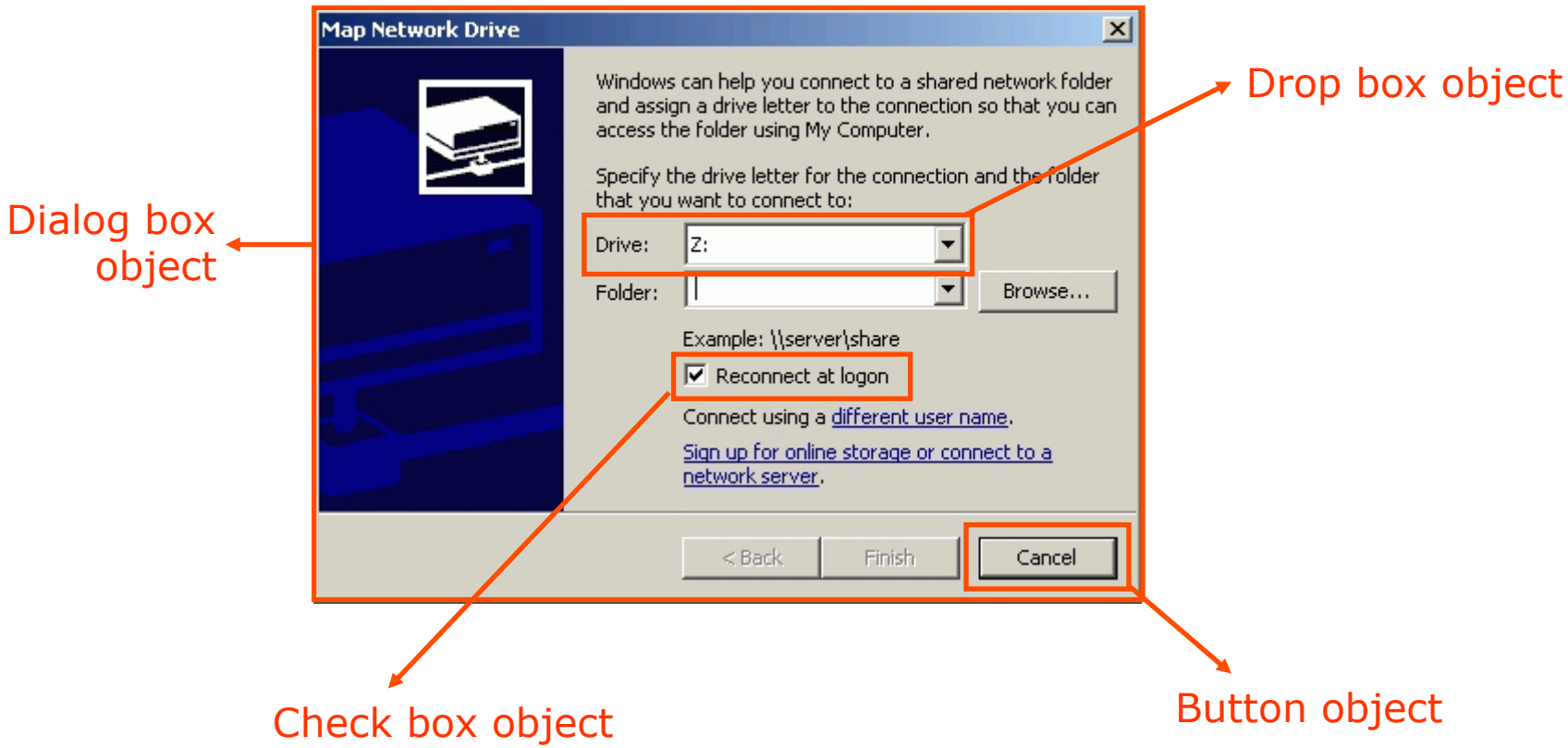
What are objects

- 'Software objects' are often found naturally in real-life problems
- Object oriented programming → Finding these objects and their role in your problem



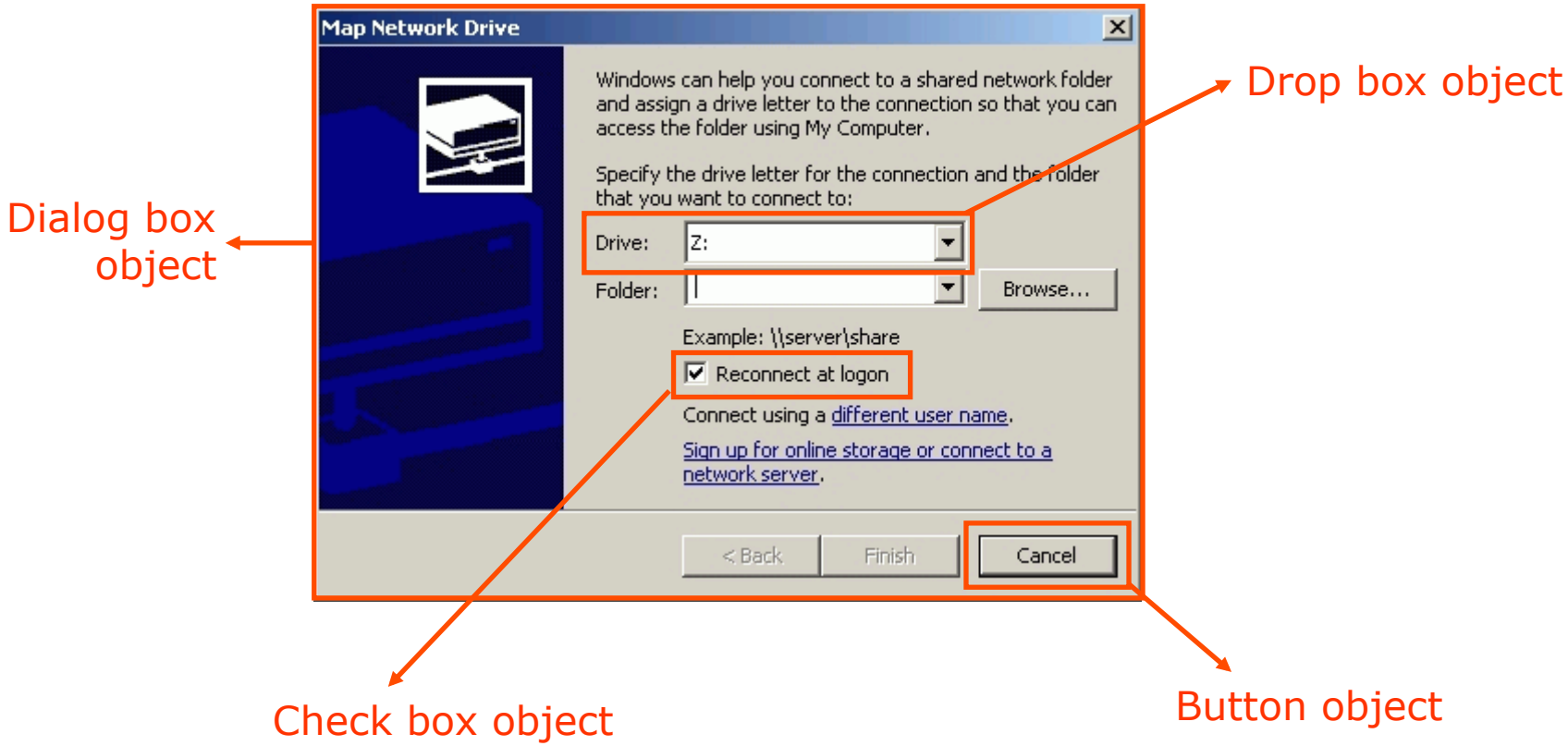
What are objects

- An object has
  - Properties : position, shape, text label
  - Behavior : if you click on the 'Cancel button' a defined action occurs



Relating objects

- Object-Oriented Analysis and Design seeks the relation between objects
  - 'Is-A' relationship (a PushButton Is-A ClickableObject)
  - 'Has-A' relationship (a DialogBox Has-A CheckBox)



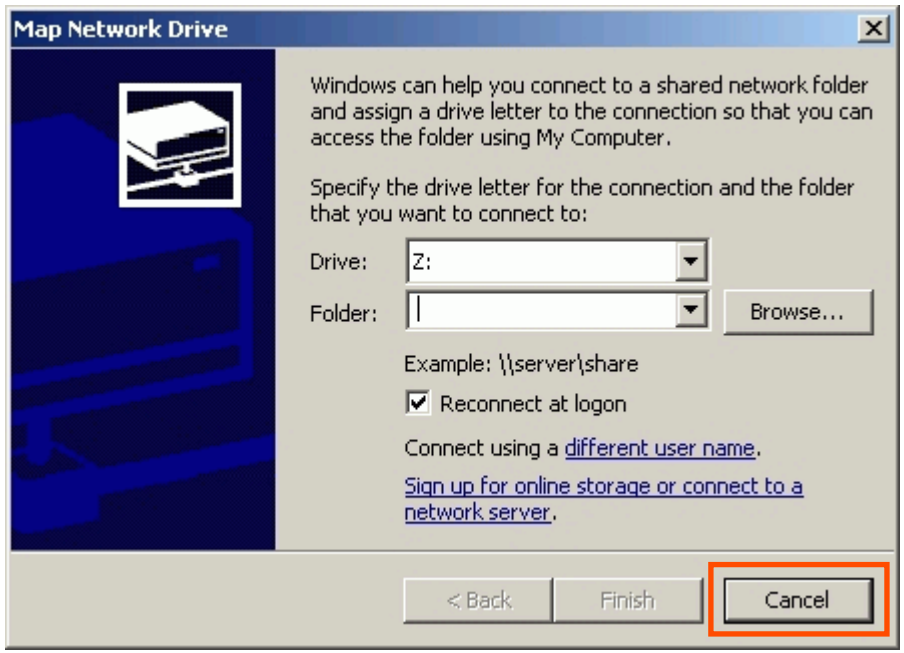
Benefits of Object-Oriented programming

- Benefits of Object-oriented programming
  - Reuse of existing code – objects can represent generic problems
  - Improved maintainability – objects are more self contained than 'subroutines' so code is less entangled
  - Often a 'natural' way to describe a system – see preceding example of dialog box
- But…
  - Object oriented modeling does not substitute for sound thinking
  - OO programming does not *guarantee* high performance, but it doesn't stand in its way either
- Nevertheless
  - *OO programming is currently the best way we know to describe complex systems*

Basic concept of OOAD

- Object-oriented programming revolves around *abstraction* of your problem.
  - Separate *what you do* from *how you do it*

- *Example – PushButton object*



PushButton is a complicated piece of software – Handling of mouse input, drawing of graphics etc..

Nevertheless you can use a PushButton object and don't need to know anything about that. Its public interface can be very simple: My name is 'cancel' and I will call function `doTheCancel()` when I get clicked

Techniques to achieve abstraction

- Abstraction is achieved through

1. Modularity

2. Encapsulation

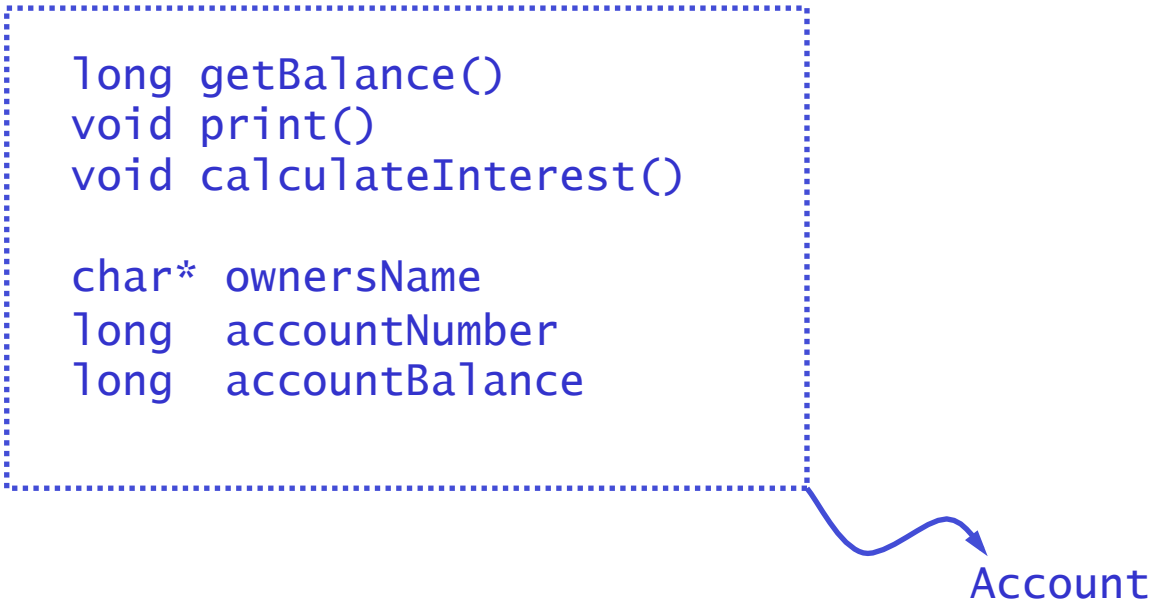
3. Inheritance

4. Polymorphism

Modularity

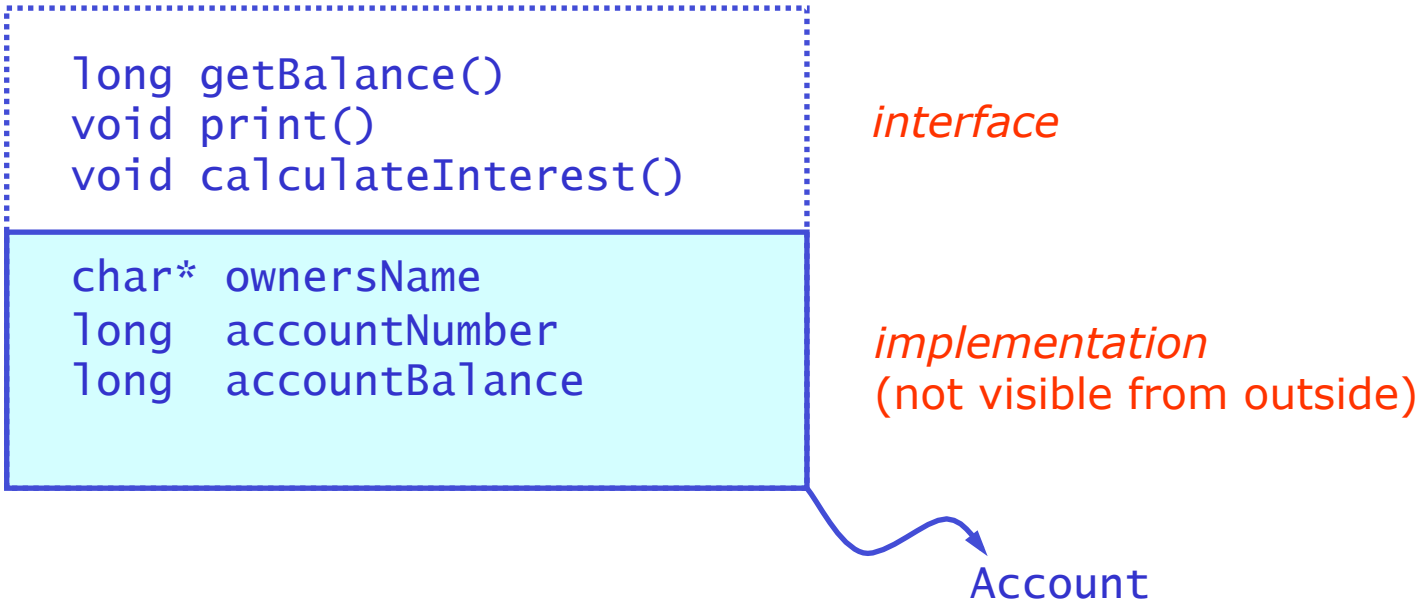
- Decompose your problem logically in independent units
  - Minimize dependencies between units – **Loose coupling**
  - Group things together that have logical connection – **Strong cohesion**

- Example
  - Grouping actions and properties of a bank account together



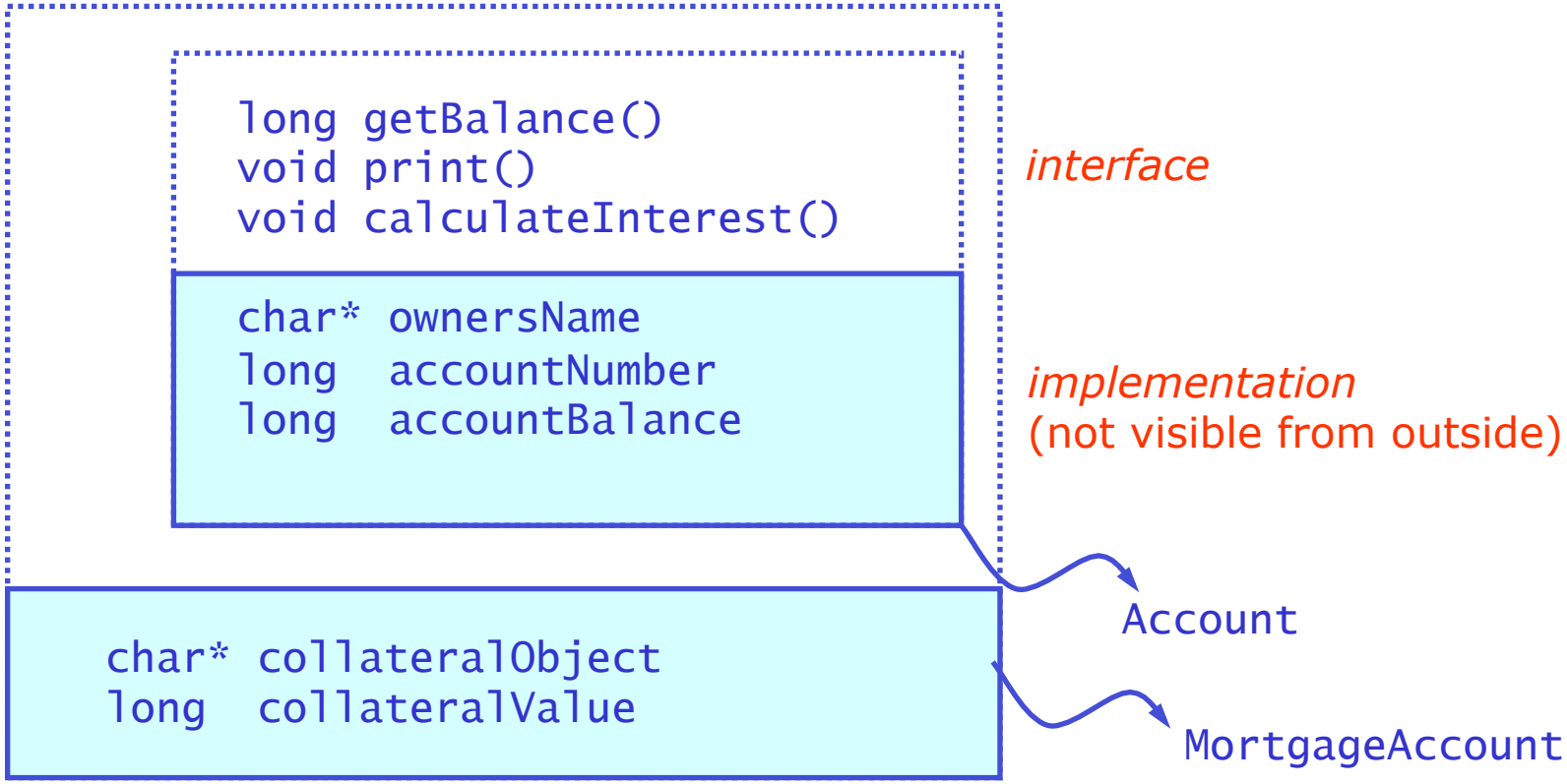
Encapsulation

- Separate interface and implementation and shield implementation from object 'users'



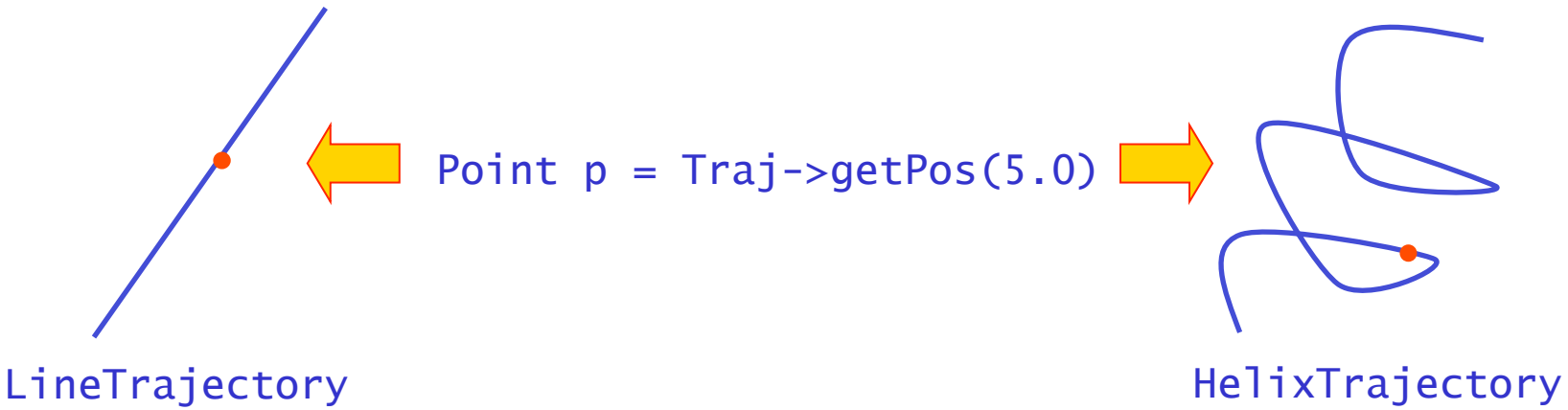
Inheritance

- Describe new objects in terms of existing objects
- Example of mortgage account



Polymorphism

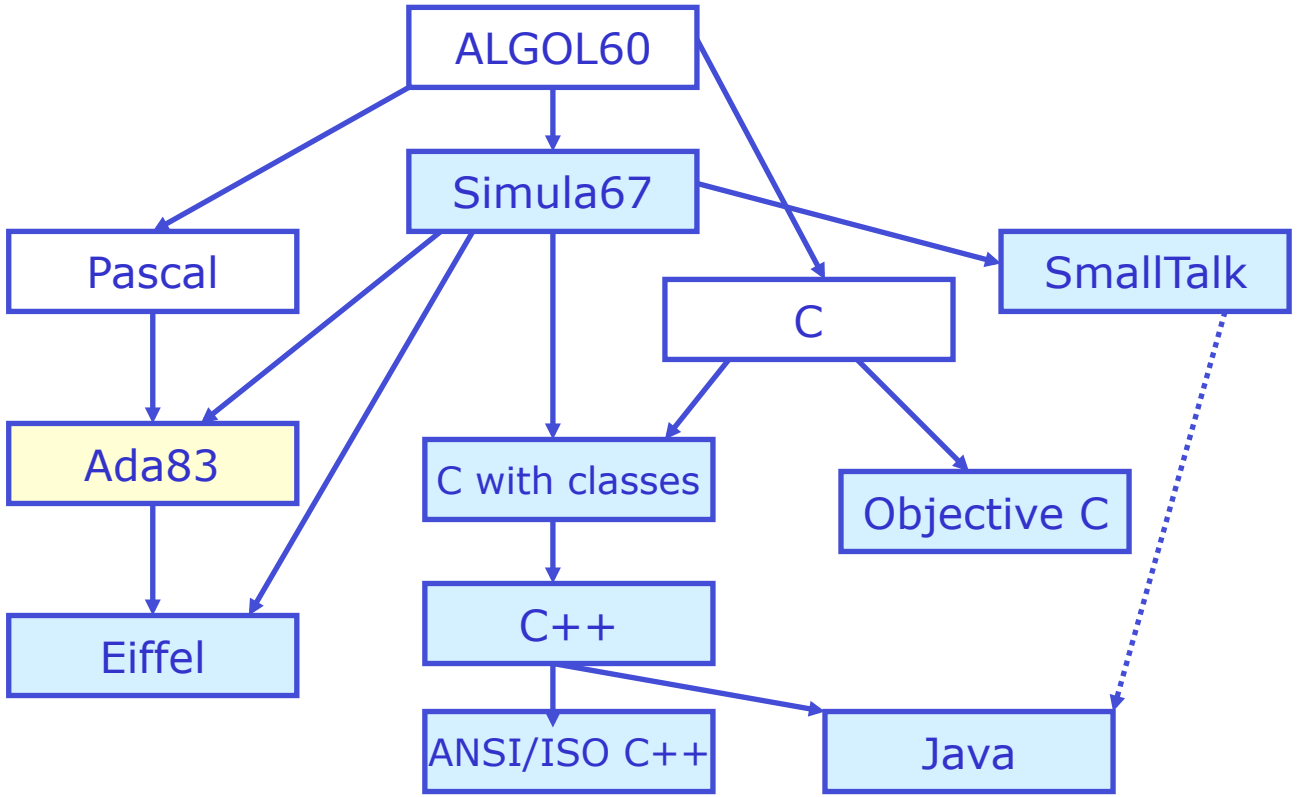
- Polymorphism is the ability to treat objects of different types the same way
  - You don't know exactly what object you're dealing with but you know that you can interact with it through a standardized interface
  - Requires some function call decisions to be taken at run time
- Example with trajectories
  - Retrieve position at a flight length of 5 cm
  - Same interface works for different objects with identical interface



Introduction to C++

- Wide choice of OO-languages – why program in C++?
  - It depends on what you need…
- Advantage of C++ – It is a compiled language
  - When used right the fastest of all OO languages
  - Because OO techniques in C++ are resolved and implemented at compile time rather than runtime so
    - **Maximizes run-time performance**
    - **You don't pay for what you don't use**
- Disadvantage of C++ – syntax more complex
  - Also, realizing performance advantage not always trivial
- C++ best used for large scale projects where performance matters
  - C++ rapidly becoming standard in High Energy Physics for mainstream data processing, online data acquisition etc…
  - Nevertheless, if your program code will be O(100) lines and performance is not critical C, Python, Java may be more efficient

C++ and other programming languages



- NB: Java very similar to C++, but simpler
  - Simpler syntax as all OO support is implemented at run-time
  - If you know C++ Java will be easy to learn

Versions of C++

- C++ is a 'living language' that evolves over time.
- This course is largely based on the 2003 standard of C++
- LHC experiments are now largely adopting C++ compilers that implement the 2011 standard of C++, which brings useful new features
  - E.g. Auto types, range-based for loops, lambdas, constructor delegation, tuples, hash tables and pointer memory management
  - I will cover a subset of these C++2011 features in this course, and explicitly point out the features that are only available in C++2011
- For the GNU compilers (gcc/g++) some of the C++2011 features are implement starting in version 4.4, with almost all features implemented in 4.7
  - In gcc 4.[3456] must add flag '-std=c++0x' to activate
  - In gcc 4.[78] must add flag '-std=c++11' to activate

Outline of the course

1. Introduction and overview
2. Basics of C++
3. Modularity and Encapsulation – Files and Functions
4. Class Basics
5. Object Analysis and Design
6. The Standard Library I – Using IOstreams
7. Generic Programming – Templates
8. The Standard Library II – The template library
9. Object Orientation – Inheritance & Polymorphism
10. Robust programming – Exception handling
11. Where to go from here