Table of contents

Design - development of something useful
Our objectives
C++ software capabilities
Goals of “good” software design
Programming approach
Design example - Doctor/Patient appointment
Doc’s information interface
Patient’s information interface
Design - step in develop. of something useful

⇒ Model(s)/Abstraction(s) to bound the problem
⇒ Real world - can’t develop in this environment
  ⇒ too complex - difficult (impossible) to include all features
  ⇒ can’t measure
  ⇒ can’t duplicate environment
  ⇒ can’t be there
  ⇒ unintended, unanticipated consequences
⇒ Model(s) - abstract some ‘features’ - ignore others

The holy grail of science/engineering is to understand the real world in order to build “better” models.
Our objective

⇒ To understand how to use C++ Objects to build better software models and systems
⇒ Objects allow users to deal with software complexity
  ⇒ Supports - encapsulation, modularity, hierarchy
⇒ Program languages are abstractions
  ⇒ Isolate user from complexities of hardware and peripheral devices
⇒ Classes - coherent view of computer’s capabilities data and functions - public or private
⇒ Enables control (action) and entities (information)
C++ software capabilities

⇒ Encapsulation - embed information and operations as needed with interfaces (public:) and protection (private:)

⇒ Modularity - classes - are self-contained and/or easily incorporate other classes on a “uses” (interface); “has a” (embed) or “is a” (inherit) basis

⇒ Hierarchy - classes can be built from simpler classes

⇒ Typing - strong definitions of elements allow the compiler to assist in attaining correct software
Goals of “good” software designs

⇒ Simple - KISS
⇒ Flexible
⇒ Accommodate change readily
⇒ Extensible
⇒ Add new features, upgrade old, etc.
⇒ Portable
⇒ Easily modified to run on numerous hosts
⇒ Reusable
⇒ Feature supported through above features
Programming approach
top/down vs bottom/up

⇒ Top down - develop the overall structure, then its constituent parts - repeat at each levels over and over again.
⇒ Bottom up - the essence is in the details. It’s the little things that count.
⇒ In complex programs both are necessary.
⇒ Both - design may have many levels. Start developing the most critical - ones that have to work. Ones that clarify the problem.
Design example - Doctor/Patient appointment

⇒ Question 1 - What does the doctor (appt. secr.) need to know about the doctor’s appointments?
⇒ Question 2 - What does the patient (appt. secr.) need to know about doctor’s availability?
⇒ Note actually there may be a single interface - appt. secr.
My Doc interface

⇒ Appointments for a specific span of dates
  6 months
⇒ Find a specific patient’s appointment(s)
⇒ Appointments for today & tomorrow
⇒ Search for a free appointment period
  give preferences - day(s), time, time span
⇒ Reserve an appointment period
⇒ Cancel an appointment period
My Patient interface

⇒ Find appointment(s)
⇒ Search for a free appointment period
give preferences - day(s), time, time span
⇒ Request reserving an appointment period
⇒ Cancel an appointment period
Doc’s information interface

⇒ Class design
Patient’s information interface

⇒ Class design