

C++ classes

- Classes are very useful — sometimes!
- Understanding when to introduce a new class in your code, and when it just makes things more confusing, requires experience
- Generally a lot of disagreements concerning best philosophies for code design
- But: When used widely, classes can at least
 - help avoid code repetition
 - help split up a task into manageable subtasks (and independent)
 - help produce more readable code
 - help with useful encapsulation / decoupling
- Don't try to turn everything into classes!
It's like talking with only nouns — there are other important concepts in the world beyond objects.
- I'm no expert on formal aspects of programming — in this course we have a pragmatic physicist "learning-by-doing" approach.
- Let's see how it works...

Terminology:

- A class: The definition of a new type
- An object: An instance of a class

MyClass mc;

The
type

↑
An instance of MyClass
(An object)

Example - 2 main.cpp

- o Look at main program first:
 - Note that code is fairly short
 - Self-explanatory, because the classes are fairly natural units and the actions they perform make sense
- o Now look at structure of the LotteryBall class
- o Show entire class body
- o Private vs Public (and Protected)
- o The private variables — defining prop of a lottery ball
- o The constructor: - What's the role?
 - Can have more than one
 - Default ctors.
- o Other methods

Recommendation: When writing a class, don't try to add all possibly useful methods right from the start. Will likely just produce a lot of unused code...

Add methods as you need them.

◦ LotteryMachine class

- A container for LotteryBalls with the ability to sample balls

- Private variable :

vector<LotteryBall>

- Public stuff:

- Two constructors

- Method~~s~~ for adding a single LotteryBall (add-ball)

- Method for creating a collection of n balls

- Method error-if-empty / demonstrate simple example of throwing / raising an error at runtime

If the error is not caught by ~~the~~ the code,
The program will stop with error message.

- Explain sample-with-replacement

sample-without-replacement

- Run code

- Show what happens when we ~~sample for~~
trigger runtime-error.

Example_3

• Code layout

include / LotteryBall.hpp
include / LotteryMachine.cpp

src / LotteryBall.cpp

src / LotteryMachine.cpp

main.cpp

• Look at main.cpp

— Nice, clean, easy to understand the basics

• Look at header files

— An interface, I can understand how I can use the class without ~~know~~ seeing the internal workings (encapsulation/decoupling)

• So I could write main.cpp just by reading the header files!

• If I want the details, I check the source files

• Look at LotteryBall.cpp and LotteryMachine.cpp

— The definitions of the methods

— Note namespace !

• Comment on include guards