Object Oriented Programming (properly done)

Paolo Burgio
paolo.burgio@unimore.it

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

High Performance Real Time Lab


Me joining
a new
company
Legacy
codebase

OOP in a ~~nutshell~~ slide

Creating abstraction of ~~real-world~~ items (***objects***) within your code, and interact with/compose them to get a goal

- Animals
- Geometrical shapes
- Timers

How to do this?

- Specify the object **class**
  - A class is a container of fields (data) and functions (code)
  - **Defines behavior** for the objects
- Create **objects** that instantiate this class
  - **Holds the status** of the single objects
- Compose them calling their methods

Class structure

You need a **constructor** and **destructor** function

- Instantiate and dispose (internal status)
- In most of languages, optional, or *implicit*

Internal fields and functions can be “protected” by marking them as **private**

- Are not accessible from outside
- As opposite to **public** (ctor, destructor, etc)

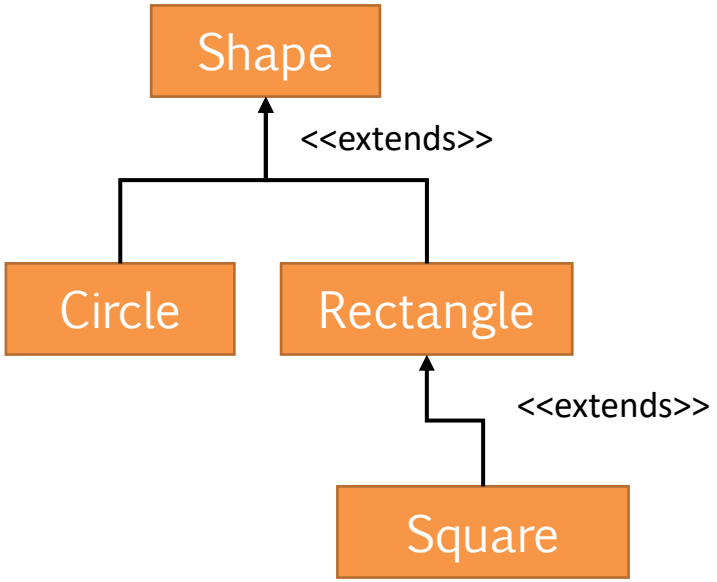
Functions can be **overloaded**

- Same name, different params
- Also, ctors can be overloaded!

Inheritance

Specify how to create your own **taxonomy** of classes

- Circle and Rectangles are Shapes
- A Square is a Rectangle as well
- They all **implement** a common “Area” functionality (specified by Shape), with slight differences
- Can **extend** base class, **overriding** specific functionalities

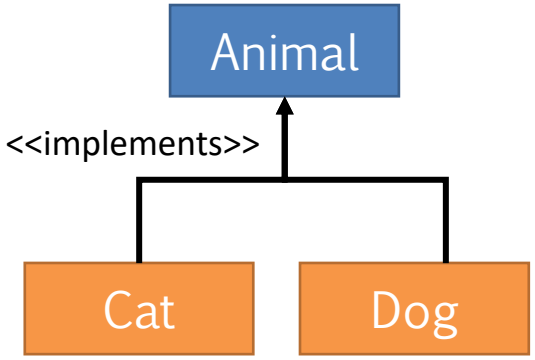


Interfaces

Interfaces are empty classes that specify a **behavior**

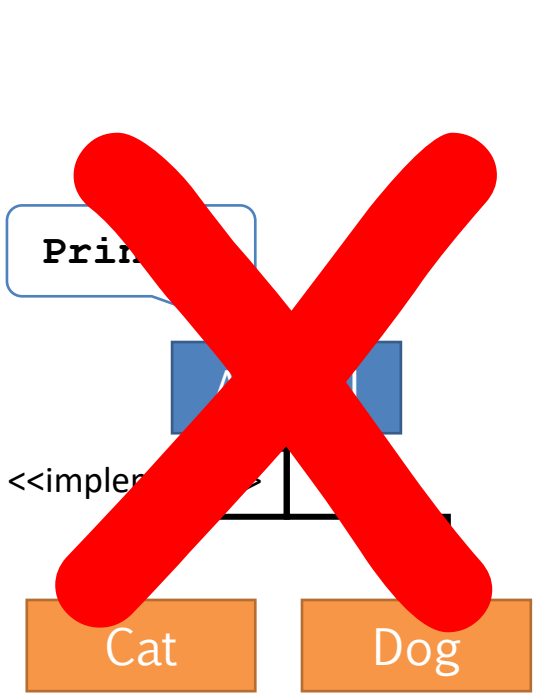
- A Cat is an Animal, but also a Dog
- Animal **declares** a "Print" functionality, that will be **implemented** by Cats and Dogs
- They let you work with an object in an opaque manner: **"Create classes, use interfaces"**

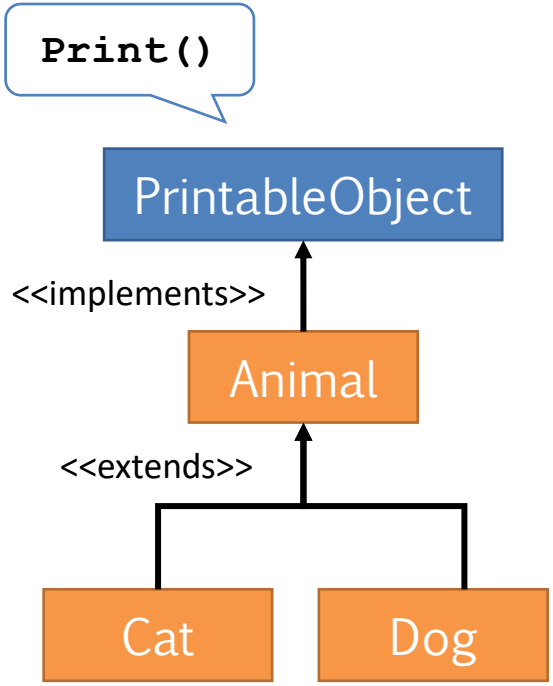
WARNING: some languages use "abstract classes", i.e., classes implemented in part



Interfaces specify behavior!

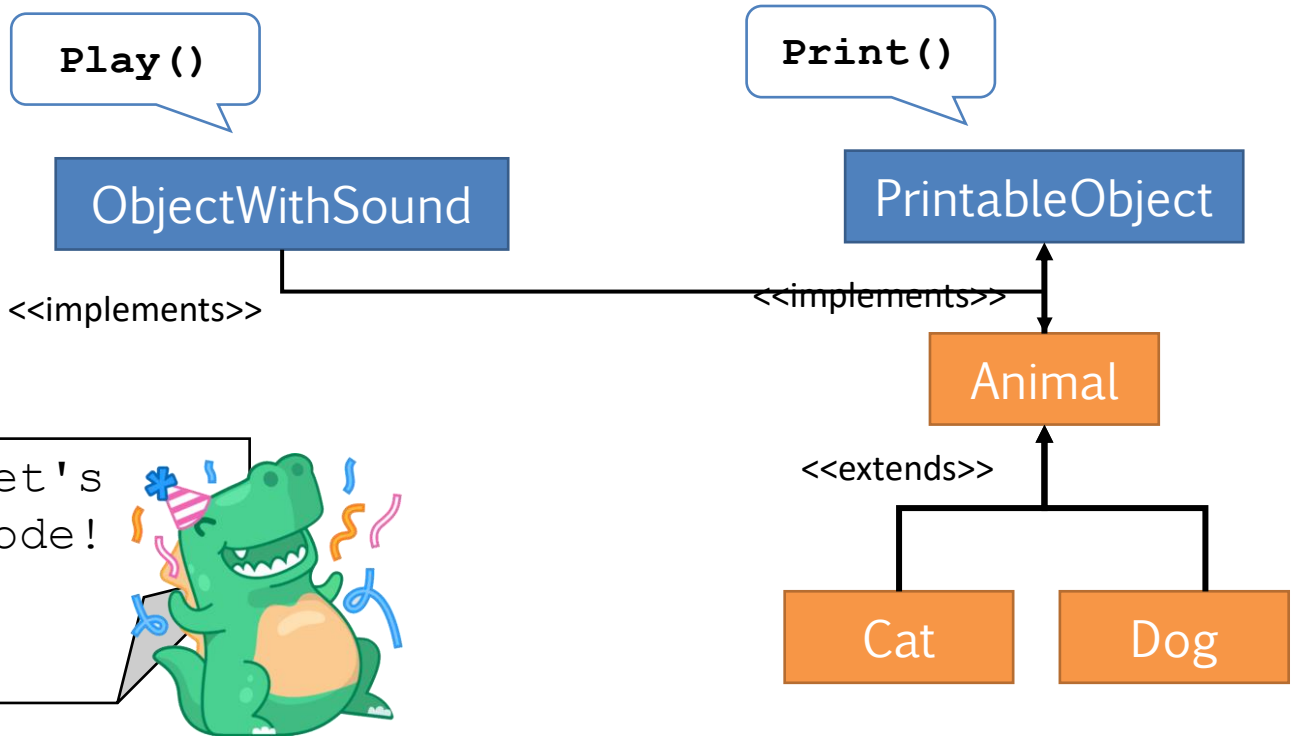
- “Animal” is not a behavior
- We actually wanted to specify a “Printable object” (behavior)
- Inherited by an “Animal” (base-class)





Multiple inheritance

- Recap: interfaces are behaviors
- Can (should) group them
- In some languages, can subclass multiple classes



The power of programming

In C++, you can do a **lot** of crazy stuff

- You can mix functional/procedural and OOP
- You can build libraries within code
- You can add thousands of classes

PLEASE

THINK BEFORE YOU CODE

Good hints

Be consistent and simple in your code!

- Use only *CamelCase* or *snake_case* (in OOP -> typically CamelCase)
- Do not mix functional programming & OOP in C++!

One class -> one file with the same name (+ header, in case)

- Unlike Microsoft..
- Create dedicated “MainClass” or “Program” class to host `main()`
- Enables portability, as it might be removed from compilation to build a library

Structure your code

- Use standard folders: `src/ include/ build/ scripts/ docs/`
- Use `cmake`
- Create `README.md` files

References

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/

Resources

- https://cliutils.gitlab.io/modern-cmake/chapters/basics.html
- Practice, practice, practice
- A "small blog"
  - http://www.google.com