CH-230-A

Programming in C and C++

C/C++

Tutorial 11

Dr. Kinga Lipskoch

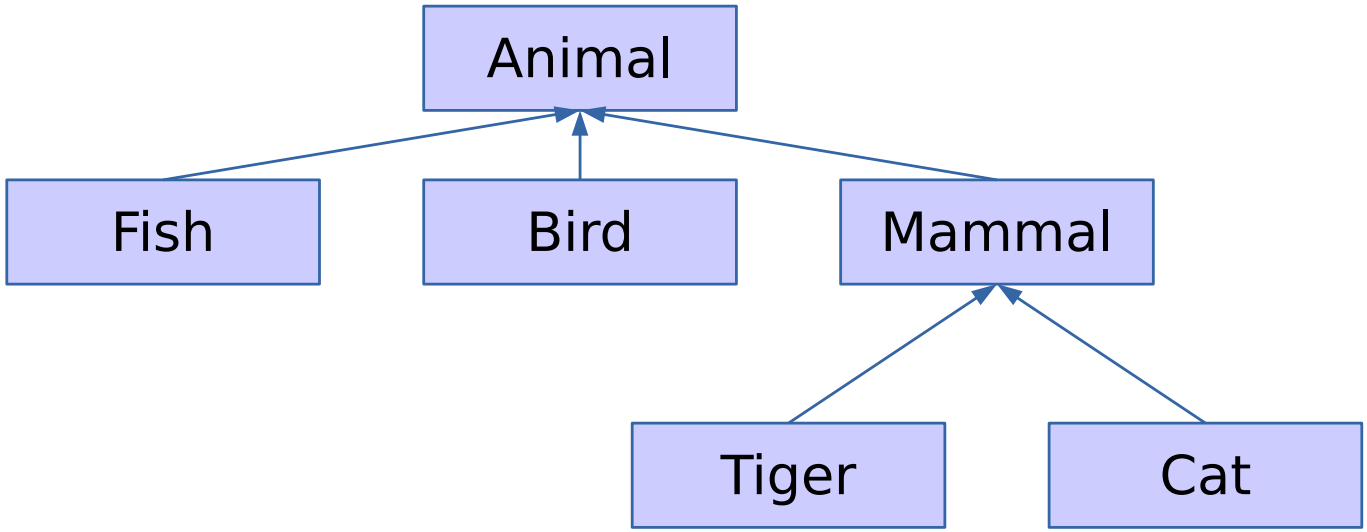
Fall 2020

Inheritance (1)

- Inheritance is used to create a new class from an existing one
  - The new (derived) class inherits what has been defined in the class (base) from which it derives
  - Great advantage in safely reusing existing code and increased abstraction capability
- Code reuse
  - Create new class as type of existing class
  - Take form of existing class and add code to it
  - Inherit from base class

Inheritance (2)

- Relationships between classes are usually graphically illustrated with trees
- Generally it is not always a tree since a node may have more than one parent, but for the moment we will consider a tree



Inheritance (3)

- Inheritance models the IS-A logic relationship between objects
  - A mammal IS-A(n) animal. Thus everything holding for an animal holds also for a mammal
    - A tiger IS-A mammal, and by transitive law, also an animal
    - Also a cat IS-A mammal, and then an animal, but different from a tiger
- To inherit means that properties in the base class are transferred into the derived class

Inheritance (4)

The IS-A relationship means that the interface of the base class is inherited by the derived class

- Thus every message which can be sent to base class can be sent to a derived class (method call)
- What is not visible in the base class will be not usable in the derived class, but will be there
  - Otherwise some information would be lost, and the interface methods could also not provide what they should

Inheritance in C++

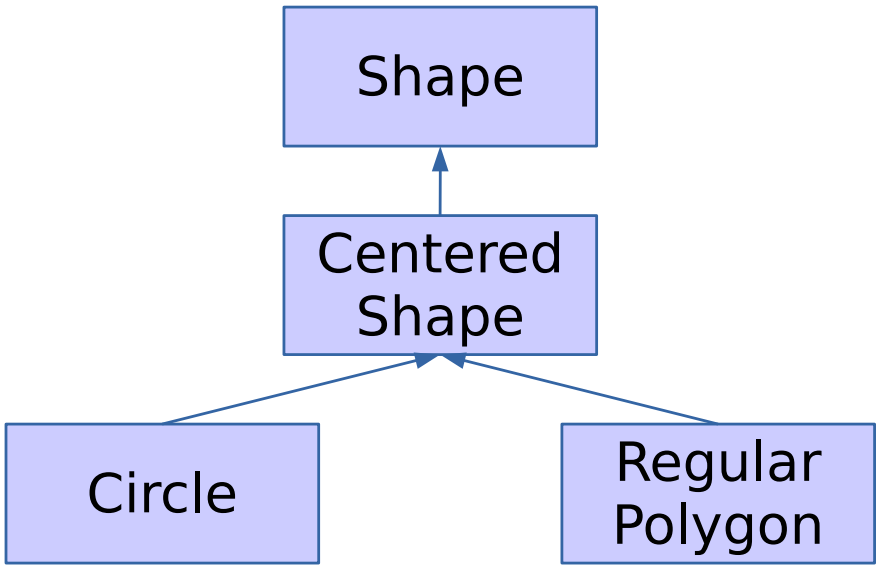
- To create a derived class, use the following syntax:

```
1 class A {
2   // whatever you need
3 };
4
5 class B : public A {
6   // B specific methods and data members
7 };
```

- Do not forget the `public` modifier, otherwise the compiler will make some not frequently used assumptions

Inheritance: An Example

- The classic shape example: devise a hierarchy of classes to model a rough drawing system
- `Shapes.h`



Initialization and Inheritance

- When initializing an object, constructors are called in sequence top to down starting from the root
- You can specify parameters in the constructor list
  Example:

```
1   B(int na, double nb): A(na) {
2      b = nb;
3   }
```

- Shapes.cpp

A Game Example

- `creature.cpp`

Inherited Interface

- `testshapes.cpp`
- On one hand, it is possible to call the method `printName` also on instances of `Circle` or `RegularPolygon`
  - Because it is part of the interface
- On the other hand, inside of `Circle` it is not possible to access `name`, because it is not part of the interface
  - The data however is there, otherwise `printName` would not work

Upcasting (1)

- As the interface of the base class is a subset of the interface of the derived class, it is possible to use a derived class where a base class is required
  - If a shape is required, then only its interface will be used. But its methods and data members are also in the interface of circle, so it is possible to use an instance of `Circle` as "`Shape`"
- Upcasting = objects are "pushed up" on the derivation tree

Upcasting (2)

- An example:

```
1   Shape *collection[2];
2   collection[0] = new Circle("A", 1, 1, 3);
3   collection[1] =
4       new RegularPolygon("B", 1, 1, 6);
5   collection[0]->printName();
6   collection[1]->printName();
```

- It works, as every message that can be sent to the base class can be sent to the derived class
- This will show its enormous power when we will deal with polymorphism

Inheritance & Destructors

- When a derived object is removed from memory, destructors are called from the leaf up to the root in the derivation tree
  - i.e., the order is reversed compared to the constructors
  - This prevents dependency problems between objects
- `simpleinheritance.cpp`

`protected`: A Third Level of Information Hiding

- We have seen that class elements can be either `public` or `private`, and what this means in terms of accessibility
- There exists a third access modifier: `protected`
- Protected means that the element is not accessible outside the class, but it is accessible in derived classes
  - It is somehow between `private` and `public`

Information Hiding: Scenario with `public` Inheritance

	Accessible outside the class	Accessible in derived class
private	no	no
protected	no	yes
public	yes	yes

There is More about Inheritance

- We have seen single inheritance. C++ allows also multiple inheritance, i.e., a class with two or more parents
  - Not always easy to use. In most cases you can do without it. Some OOP languages (like Java) do not even provide multiple inheritance
    - We will discuss multiple inheritance later (example)
- We have seen `public` inheritance: it is also possible to specify `private` or `protected` inheritance
- See 13.6 in C++ Annotations or Eckel's book for details

`protected` and `private` Inheritance

Derivation	Access rights in base class	Access rights in derived class
public	public	public
	protected	protected
	private	not accessible
private	public	private
	protected	private
	private	not accessible
protected	public	protected
	protected	protected
	private	not accessible

Summary

- A derived class is created by
  `class derivedclass : public baseclass`
- A derived class has all data of its direct and indirect base classes
- A derived class can directly access data and methods which are declared as `public` or `protected`
- If the base class does not have a default constructor, a derived class must call a constructor of the base class explicitly