Objects in C++

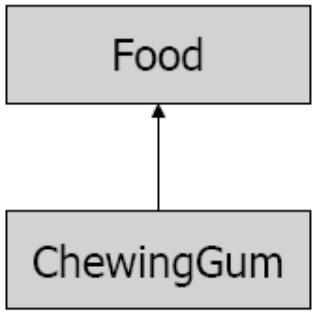
Inheritance

C++ Object System

- Object-oriented features
  1. Classes and Data Abstraction
  2. Encapsulation
  3. Inheritance
     - Single and multiple inheritance
     - Public and private base classes
  4. Objects, with dynamic lookup of virtual functions
  5. Subtyping
     - Tied to inheritance mechanism

Inheritance (1)

The ability to reuse the definition of one kind of object to define another kind of object.

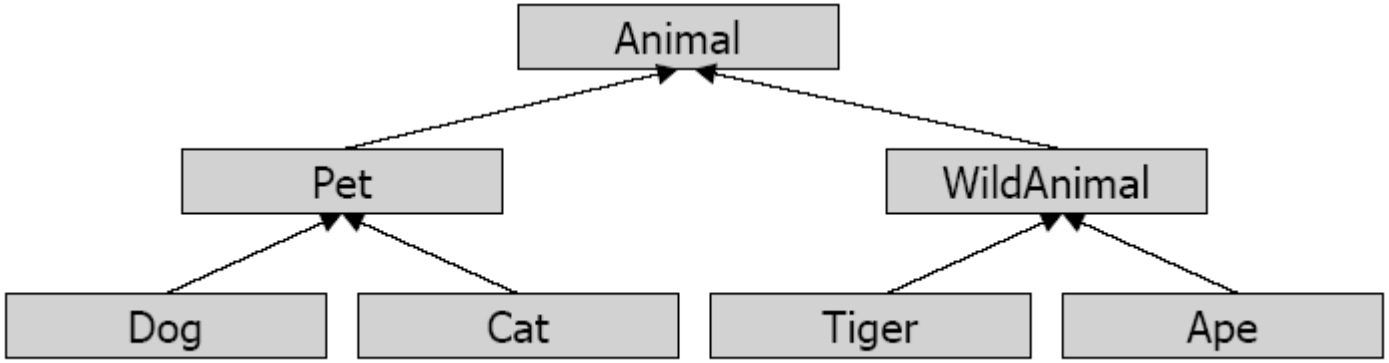



```
class ChewingGum : public Food {
  // ...
};
```

- ChewingGum inherits
  - all public class members (full access)
  - all protected class members (full access)
  - all private class members (no direct access)

Class hierarchies

- each derived class can act as a base class for further derivation



Constructors/destructors and inheritance (1)

- constructors
  - require calling the base class constructor
  - if arguments are mandatory, they have to be provided

```
class Manager : public Employee {
  public:
   Manager(const std::string& name,
                const short level)
         : Employee(name), level_(level) {
   }
  private:
   short level_;
};
```

Constructors/destructors and inheritance (2)

- destructors
  - always make destructors virtual in base classes
  - there might be cleanup work to be done in derived classes

```
class Employee {
  //...
  public:
   //...
   virtual ~Employee() {}
};
```

Public, private, protected inheritance

```
class CD: public CB{…}
class CD: private CB{…} or class CD: CB{…}
class CD: protected CB{…}
```

		TIPO di EREDITARIETA'		
		public	protectet	private
VISIBILITA'	public	public	protected	private
	protected	protected	protected	private
	private	private	private	private

Private inheritance –publicize members

```
class CBase {
   int x;
public:
   int y;
   voif f();
   void f(int);
};
class CDerivata: Cbase{ // private inheritance
public:
   CBase::y; // y is turned in pubblic
   CBase::x; // ERROR. Not allowed!! x is private
   CBase::f; // Both overloaded members exposed
};
```

- Thus, **private** inheritance is useful if you want to hide part of the functionality of the base class.
- In the presence of private inheritance, a subclass in not a subtype

Multiple inheritance

- simply extend the inheritance definition

```
class MobileAgentCommand :
      public Command,
      public Serializable,
      public PersistentObject {

   ...
};
```

However, multiple inheritance introduces a number of possibilities for ambiguity!

Redefining (1)

```
class X {
  int i;
  public:
  X() { i = 0; }
  void set(int ii) { i = ii; }
  int  permute() { return i = i * 47; }
};

class Y : public X {
 int i; // Different from X's i
 public:
 Y() { i = 0; }
 int change() {
   i = permute(); // Different name call
   return i;
 }
 void set(int ii) {  // redefining
   i = ii;
   X::set(ii); // Same-name function call
 }
};
```

Redefining (2)

- *Redefining* for ordinary member functions and *overriding* when the base class member function is a **virtual** function
- *Redefining* **produces an overloaded function, with code selection done at compile time** through the operator *class_name::*
- **Virtual** functions are the normal case and will be covered in detail later
- **Polymorphism** is implemented in C++ with the **dynamic lookup of virtual functions**

Redefining (3)

```
#include <iostream>
class A{
 int i;
 public:
 A(): i(1){};
 int f(){ return i;}
};
class B: public A{
 int i;
 public:
 B():i(2){};
 void f(int s){i = s;} //REDEFINING
 int g(){
      // return f(); ERROR
      return A::f(); //OK
 }
};
```