

Chapter 7. Inheritance and Classes

Programming Concepts in Scientific
Programming

EPFL, Master class

October 28, 2019

Polymorphism

```
class Animal {  
  
public:  
    void scream() {  
        std::cout << name;  
        std::cout << ": AAAAAAA" << std::endl;  
    }  
  
private:  
    std::string name;  
};
```

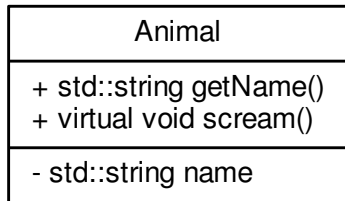
Polymorphism

```
class Animal {  
  
public:  
    void scream() {  
        std::cout << name;  
        std::cout << ": AAAAAAA" << std::endl;  
    }  
  
private:  
    std::string name;  
};
```

Animal
+ std::string getName() + virtual void scream()
- std::string name

Polymorphism

UML



Unified Modeling Language (UML)

Polymorphism

```
class Cat {  
public:  
    void scream() { std::cout << name << ": MIAOU\n"; }  
  
private:  
    std::string name;  
};
```

Polymorphism

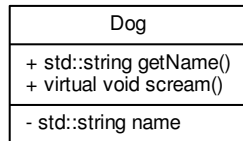
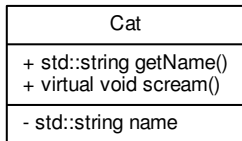
```
class Cat {
public:
    void scream() { std::cout << name << ": MIAOU\n"; }

private:
    std::string name;
};

class Dog {
public:
    void scream() { std::cout << name << ": OUAFF\n"; }

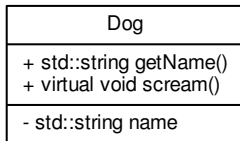
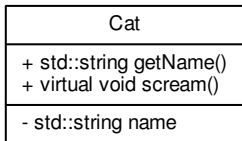
private:
    std::string name;
};
```

Polymorphism



Unified Modeling Language (UML)

Polymorphism



Unified Modeling Language (UML)

Lot of duplicated code

Can I write this ?

```
Cat list[2];  
Cat c;  
Dog d;  
list[0] = c;  
list[1] = d;
```

Polymorphism

```
class Animal {  
  
public:  
    void scream() {  
        if (type == "cat") {  
            std::cout << name << ": MIAOU\n";  
        }  
        if (type == "dog") {  
            std::cout << name << ": OUAFF\n";  
        }  
    }  
}
```

Polymorphism

```
class Animal {  
  
public:  
    void scream() {  
        if (type == "cat") {  
            std::cout << name << ": MIAOU\n";  
        }  
        if (type == "dog") {  
            std::cout << name << ": OUAFF\n";  
        }  
    }  
}
```

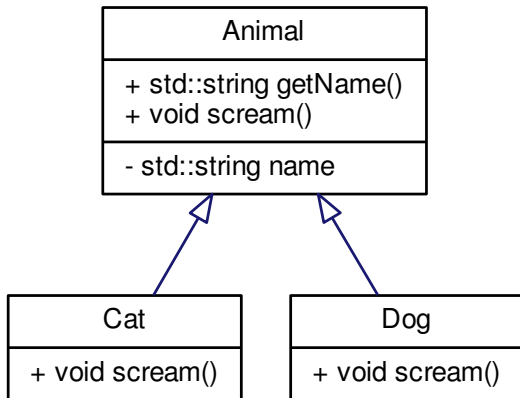
If there are many animals:
long list of "if statements"

Solution ?

Solution ?

Inheritance

Inheritance



Inheritance

```
class Animal {  
public:  
    void scream() { std::cout << name << ": AAAAA\n"; }  
  
protected:  
    std::string name;  
};
```

Inheritance

```
class Animal {  
public:  
    void scream() { std::cout << name << ": AAAAA\n"; }  
  
protected:  
    std::string name;  
};
```

```
class Cat : public Animal {  
public:  
    void scream() { std::cout << name << ": MIAOU\n"; }  
};
```


Inheritance

```
class Animal {  
public:  
    void scream() { std::cout << name << ": AAAAA\n"; }  
  
protected:  
    std::string name;  
};
```

```
class Cat : public Animal {  
public:  
    void scream() { std::cout << name << ": MIAOU\n"; }  
};
```

```
class Dog : public Animal {  
public:  
    void scream() { std::cout << name << ": OUAFF\n"; }  
};
```

Inheritance

Encapsulation: protected

```
class Animal {  
public:  
    void scream() { std::cout << name << ": AAAAA\n"; }
```

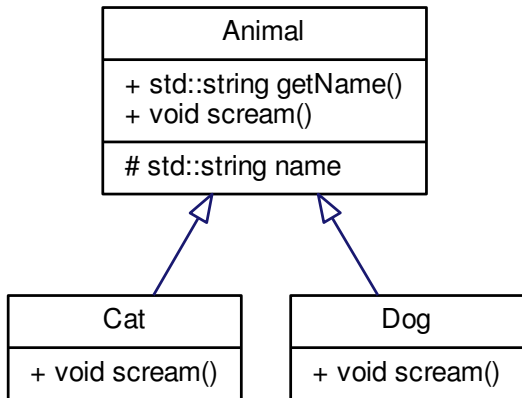
protected:

```
    std::string name;  
};
```

```
class Cat : public Animal {  
public:  
    void scream() { std::cout << name << ": MIAOU\n"; }
```

protected: permission to all daughter classes

Class diagram



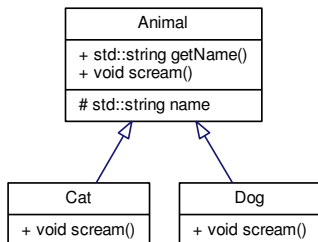
Polymorphism & Casting

Now I can do this:

```
void casting() {  
    Cat c;  
    Animal *ptr = &c;  
    ptr->scream();  
}
```

Or this:

```
void makeItScream(Animal &a) {  
    // do not know if 'a'  
    // is Cat or Dog  
    a.scream();  
}
```



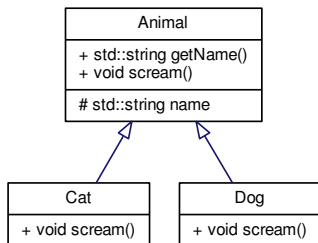
Polymorphism & Casting

Now I can do this:

```
void casting() {  
    Cat c;  
    Animal *ptr = &c;  
    ptr->scream();  
}
```

Or this:

```
void makeItScream(Animal &a) {  
    // do not know if 'a'  
    // is Cat or Dog  
    a.scream();  
}
```



► Which **scream** method is called ?

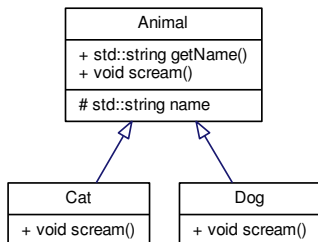
Polymorphism & Casting

Now I can do this:

```
void casting() {  
    Cat c;  
    Animal *ptr = &c;  
    ptr->scream();  
}
```

Or this:

```
void makeItScream(Animal &a) {  
    // do not know if 'a'  
    // is Cat or Dog  
    a.scream();  
}
```



- ▶ Which **scream** method is called ?
- ▶ Which **scream** method should be called ?

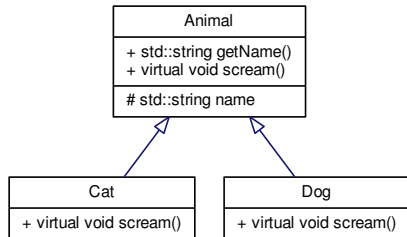
Polymorphism

```
class Animal {  
  
public:  
    virtual void scream() {  
        std::cout << name;  
        std::cout << ": AAAAA\n";  
    }  
}
```

Polymorphism

```
class Animal {  
  
public:  
    virtual void scream() {  
        std::cout << name;  
        std::cout << ": AAAAA\n";  
    }  
}
```

- ▶ **virtual** is important: brings polymorphism
- ▶ If forgotten: broken polymorphism



Modern C++ (not in the book)

► Mother class

```
class Animal {  
public:  
    virtual void scream() { std::cout << name << ": AAAAA\n"; }  
  
protected:  
    std::string name;  
};
```

► Daughter class

```
class Dog : public Animal {  
public:  
    void scream() override {  
        Animal::scream();  
        std::cout << name << ": OUAFF\n";  
    }  
};
```

Calling mother methods

```
public:  
    void scream() override {  
        Animal::scream();  
        std::cout << name << ": OUAFF\n";  
    }  
};
```

Calling mother constructor

```
class Dog : public Animal {  
  
public:  
    Dog(std::string &name) : Animal(name) {}  
  
    void scream() override {  
        Animal::scream();  
        std::cout << name << ": OUAFF" << std::endl;  
    }  
};
```

Abstract class

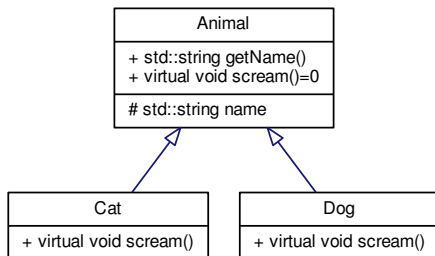
What if we want no default to a mother's method ?

```
class Animal {  
  
public:  
    Animal(std::string name) { this->name = name; }  
    virtual void scream() = 0;  
  
protected:  
    std::string name;  
};
```

Abstract class

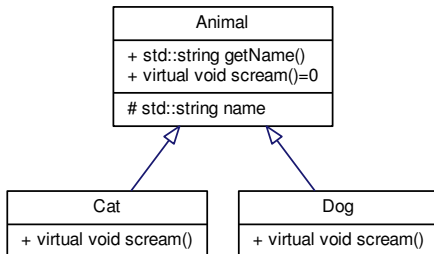
What if we want no default to a mother's method ?

```
class Animal {  
  
public:  
    Animal(std::string name) { this->name = name; }  
    virtual void scream() = 0;  
  
protected:  
    std::string name;  
};
```



Abstract class

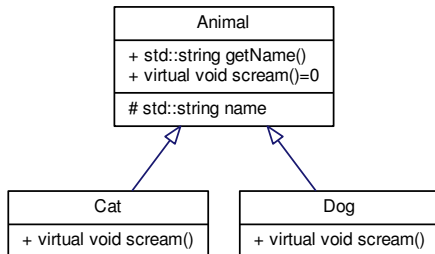
What if we want no default to a mother's method ?



- ▶ `scream` is a **pure virtual** function.

Abstract class

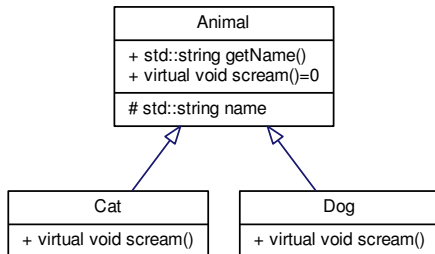
What if we want no default to a mother's method ?



- ▶ `scream` is a **pure virtual** function.
- ▶ This makes *Animal* an abstract class.

Abstract class

What if we want no default to a mother's method ?



- ▶ `scream` is a **pure virtual** function.
- ▶ This makes *Animal* an abstract class.
- ▶ Abstract classes can provide object oriented **interfaces**

Abstract class

Cannot instantiate an object of an abstract class

```
// this will lead to compilation error
```

```
Animal a("kitty");
```

```
Cat c("kitty"); // ok
```

```
Animal *ptr = &c; // ok
```

```
Animal &ref = c; // ok
```

Take away message

- ▶ **UML diagrams:** represent classes associations
- ▶ **Inheritance:** share **members** and **methods** between **mother** class and **daughter** classes
- ▶ **Inheritance** allows **code factorization**
- ▶ **protected** methods/members are private to the world but public to **daughter** classes
- ▶ **virtual methods:** methods declared as **polymorphic**
- ▶ **pure virtual methods:** methods declared as **polymorphic** with no default behavior
- ▶ **Abstract class** has at least one pure virtual method and is **incomplete:** cannot be instantiated.
- ▶ **Polymorphism:** allows to manipulate cats and dogs as generic animals