Chapter 7. Inheritance and Classes

Programming Concepts in Scientific Computing

EPFL, Master class

October 18, 2023

Polymorphism

```cpp
class Animal {

public:
  void scream() {
    std::cout << name;
    std::cout << ": AAAAAAA" << std::endl;
  }

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

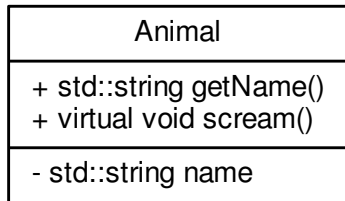# Polymorphism

```cpp
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

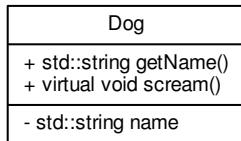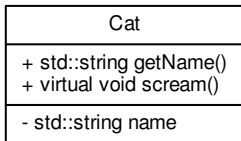| Animal |
| --- |
| + std::string getName()
+ virtual void scream() |
| - std::string name |

Unified Modeling Language (UML)

Polymorphism

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

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

Polymorphism

```cpp
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

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

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

Unified Modeling Language (UML)

Polymorphism

Unified Modeling Language (UML)

Lot of duplicated code

Can I write this ?

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

Polymorphism

```cpp
class Animal {

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

private:
```

Polymorphism

```cpp
class Animal {

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

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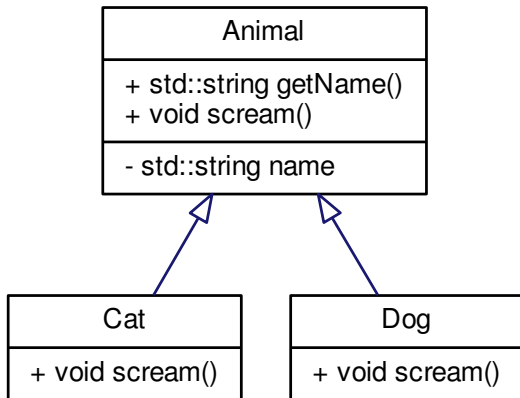
If there are many animals:
long list of "if statements"

Solution ?

Solution ?

Inheritance

Inheritance

Animal

+ std::string getName()
+ void scream()

- std::string name

Cat

+ void scream()

Dog

+ void scream()

Inheritance

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

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

Inheritance

```cpp
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

```cpp
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"; }
};
```
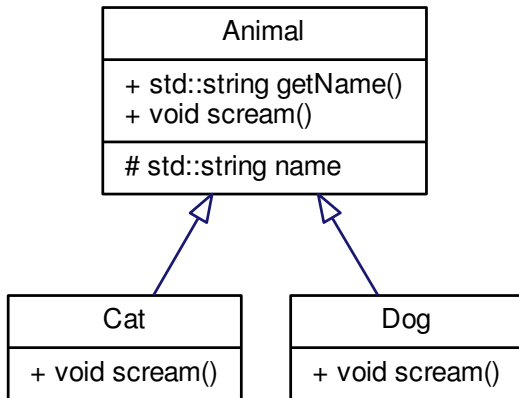
Inheritance

```
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

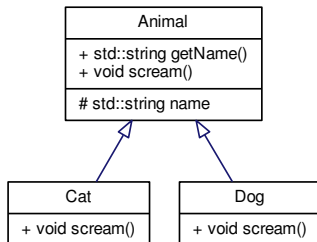# Class diagram

Polymorphism & Casting

Now I can do this:

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

Or this:

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

Polymorphism & Casting

Now I can do this:

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

Or this:

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


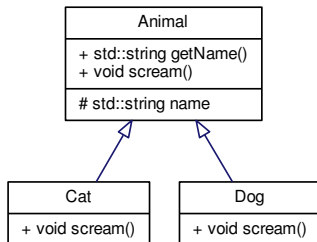- ▶ Which **scream** method is called ?

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();
}
```


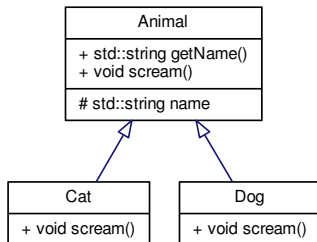- ▶ Which **scream** method is called ?

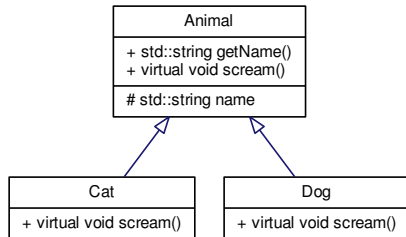- ▶ Which **scream** method should be called ?

Polymorphism

```cpp
class Animal {

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

Polymorphism

```cpp
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

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

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

▶ Daughter class

```cpp
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

```cpp
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;
};
```
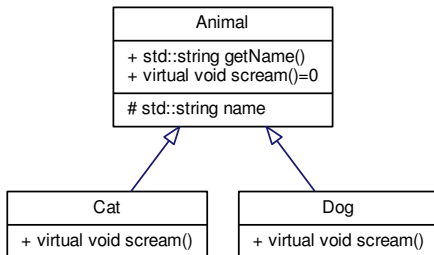
Abstract class

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

```cpp
class Animal {

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

protected:
  std::string name;
};
```
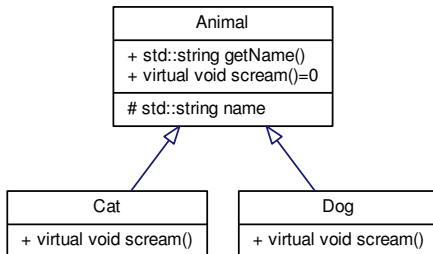
Abstract class

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


```
                    ┌─────────────────────────────┐
                    │          Animal             │
                    ├─────────────────────────────┤
                    │ + std::string getName()     │
                    │ + virtual void scream()=0   │
                    ├─────────────────────────────┤
                    │ # std::string name          │
                    └─────────────────────────────┘
```

```
┌─────────────────────────┐   ┌─────────────────────────┐
│          Cat            │   │          Dog            │
├─────────────────────────┤   ├─────────────────────────┤
│ + virtual void scream() │   │ + virtual void scream() │
└─────────────────────────┘   └─────────────────────────┘
```

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

Abstract class

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


```
                        Animal
            + std::string getName()
            + virtual void scream()=0
            # std::string name
```

```
        Cat                       Dog
+ virtual void scream()    + virtual void scream()
```
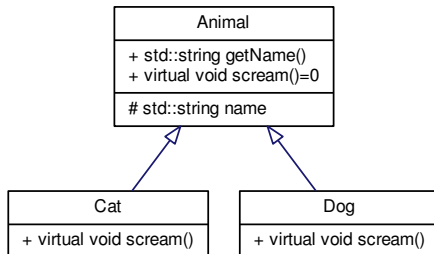
▶ scream is a **pure virtual** function.

▶ This makes *Animal* an abstract class.
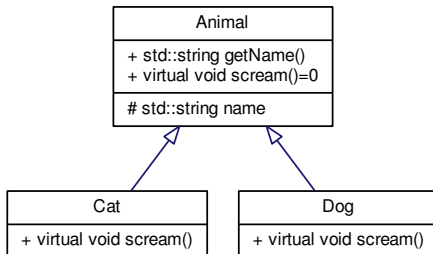
Abstract class

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

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

Abstract class

Cannot instantiate an object of an abstract class

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

// Can manipulate with daugther class reference
Animal *ptr = &c; // ok
Animal &ref = c;  // ok
```

Abstract class

Cannot instantiate an object of an abstract class

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

// Can manipulate with daugther class reference
Animal *ptr = &c; // ok
Animal &ref = c;  // ok

makeItScream(c); // polymorphism
}

void makeItScream(Animal &a) { a.scream(); }
```

Inheritance
public, protected and private

public inheritance

public base members $\rightarrow$ public in derived
protected base members $\rightarrow$ protected in derived.

protected inheritance

public&protected base members $\rightarrow$ protected in derived

private inheritance

public&protected base members $\rightarrow$ private in derived

Inheritance
public, protected and private

public inheritance
public base members → public in derived
protected base members → protected in derived.

protected inheritance
public&protected base members → protected in derived

private inheritance
public&protected base members → private in derived

*see **private_inheritance.hh** in code snippets*

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 nethod and is **incomplete**: cannot be instanciated.
- **Polymorphism**: allows to manipulate cats and dogs as generic animals