Object-Oriented Concept

- **Encapsulation**
  - ADT, Object

- **Inheritance**
  - Derived object

- **Polymorphism**
  - Each object knows what it is
Polymorphism

- noun, the quality or state of being able to assume different forms - Webster
- An essential feature of an OO Language
- It builds upon Inheritance
Polymorphism

class Dog {
    public:
        virtual void bark() = 0;
};
class 불독 : public Dog {
    public:
        void bark() {std::cout << "왈왈" << std::endl;}
};
class 진돗개 : public Dog {
    public:
        void bark() {std::cout << "멍멍" << std::endl;}
};
class 치와와 : public Dog {
    public:
        void bark() {std::cout << "댕댕" << std::endl;}
};

CLIENT CODE

void do_bark(Dog *dog) {
    dog->bark();
}

int main() {
    불독 Bulldog;
    진돗개 Jindog;
    치와와 Chiwawa;

do_bark(&Bulldog);
do_bark(&Jindog);
do_bark(&Chiwawa);
}

OUTPUT

왈왈
멍멍
댕댕
Static Binding

• When the type of a formal parameter is a parent class, the argument used can be:
  
  the same type as the formal parameter,
  
or,
  
  any derived class type.

• Static binding is the compile-time determination of which function to call for a particular object based on the type of the formal parameter.

• When pass-by-value is used, static binding occurs.
Dynamic Binding

- Is the **run-time determination** of which function to call for a particular object of a derived class based on the type of the argument.

- Declaring a member function to be **virtual** instructs the compiler to generate code that guarantees dynamic binding.

- Dynamic binding requires **pass-by-reference**.
Virtual Functions

- Virtual Functions overcome the problem of run time object determination

- Keyword `virtual` instructs the compiler to use late binding and delay the object interpretation

- How?
  - Define a virtual function in the base class. The word `virtual` appears only in the base class
  - If a base class declares a virtual function, it must implement that function, even if the body is empty
  - Virtual function in base class stays virtual in all the derived classes
  - It can be overridden in the derived classes
  - But, a derived class is not required to re-implement a virtual function. If it does not, the base class version is used
Pure Virtual Function

class Dog {
    public:
    virtual void bark() = 0;
    void bark() {std::cout << "??" << std::endl;}
};
class 불독 : public Dog {
    public:
    void bark() {std::cout << "왈왈" << std::endl;}
};
class 진돗개 : public Dog {
    public:
    void bark() {std::cout << "멍멍" << std::endl;}
};
class 치와와 : public Dog {
    public:
    void bark() {std::cout << "깅 Kling" << std::endl;}
};
Virtual Function

class Dog {
    public:
        virtual void bark() = 0;
};
class 불독 : public Dog {
    public:
        void bark() {std::cout << "왈왈" << std::endl;}
};
class 진돗개 : public Dog {
    public:
        void bark() {std::cout << "멍멍" << std::endl;}
};
class 치와와 : public Dog {
    public:
        void bark() {std::cout << "จอง댕" << std::endl;}
};

CLIENT CODE

int main() {
    Dog *dog = new Jindog;
    dog->bark();
    치와와 Chiwawa;  Dynamic Binding
    dog = &Chiwawa;
    dog->bark();
}

OUTPUT

멍멍
왈왈
Virtual Destructor

class Dog {
  public:
    virtual void bark() = 0;
};
class 불독 : public Dog {
  public:
    void bark() {std::cout << “왈왈” << std::endl;}
};
class 진돗개 : public Dog {
  public:
    void bark() {std::cout << “멍멍” << std::endl;}
};
class 치와와 : public Dog {
  public:
    void bark() {std::cout << “괭괭” << std::endl;}
};

CLIENT CODE

int main() {
  Dog *a = new 진돗개;
  진돗개 *b = new 진돗개;
  delete a;
  delete b;
}
delate a:
Call only 진돗개’s destructor
delete b:
Call 진돗개’s destructor &
Dog’s destructor
Virtual Destructor

class Dog {
    public:
        virtual void bark() = 0;
    ~Dog() { };
};
class 불독 : public Dog {
    public:
        void bark() { std::cout << "왈왈" << std::endl; }
};
class 진돗개 : public Dog {
    public:
        void bark() { std::cout << "멍멍" << std::endl; }
};
class 치와와 : public Dog {
    public:
        void bark() { std::cout << "깽깽" << std::endl; }
};

CLIENT CODE

int main() {
    Dog *a = new 진돗개;
    진돗개 *b = new 진돗개;
    delete a;
    delete b;
}
Summary

- When you use virtual functions, the compiler stores additional information about the types of objects available and created.

- Polymorphism is supported at this additional overhead.

- Important:
  - Virtual functions work only with pointers/references.
  - Not with objects even if the function is virtual.
  - If a class declares any virtual methods, the destructor of the class should be declared as virtual as well.
Area of the figure

- Input
  - Type of figure (triangle or rectangle)
  - Point
- Output: area for Rectangle

$ ./figure
Figure: triangle
Point1: 0,0
Point2: 2,0
Point3: 2,2
Area: 2