

Поліморфізм

Слово поліморфізм означає "мати багато форм".

Поліморфізм можна продемонструвати більш наочно на прикладі: припустимо, ви хочете зробити просту гру, яка включає різних ворогів: монстрів, ніндзя тощо. У всіх ворогів є одна спільна функція: `attackPower`. Однак вони атакують по-різному. У цій ситуації поліморфізм дозволяє викликати одну і ту ж функцію нападу на різні об'єкти, але призводить до різної поведінки.

```
class Enemy {  
protected:  
    int attackPower;  
public:  
    void setAttackPower(int a){  
        attackPower = a;  
    }  
};
```

```
class Ninja: public Enemy {  
public:  
    void attack() {  
        cout << "Ninja! - " << attackPower << endl;  
    }  
};  
  
class Monster: public Enemy {  
public:  
    void attack() {  
        cout << "Monster! - " << attackPower << endl;  
    }  
};
```

```
int main() {  
    Ninja n;  
    Monster m;  
}
```

Ninja та Monster успадковуються від Enemy, тому всі об'єкти Ninja та Monster є об'єктами Enemy. Це дозволяє нам зробити наступне:

```
Enemy *e1 = &n;  
Enemy *e2 = &m;
```

Зараз ми створили два покажчики типу Enemy, вказуючи їх на об'єкти Ninja та Monster.

Тепер ми можемо викликати відповідні функції:

```

int main() {
    Ninja n;
    Monster m;
    Enemy *e1 = &n;
    Enemy *e2 = &m;

    e1->setAttackPower(20);
    e2->setAttackPower(80);

    n.attack();
    m.attack();
}

/* Output:
Ninja! - 20
Monster! - 80
*/

```

Virtual Functions

Щоб мати можливість викликати відповідну функцію `attack()` для кожного з похідних класів за допомогою покажчиків `Enemy`, нам потрібно оголосити функцію базового класу як **virtual**.

Визначення **віртуальної функції** в базовому класі з відповідною версією у похідному класі дозволяє поліморфізму використовувати вказівники `Enemy` для виклику функцій похідних класів.

Кожен похідний клас замінить функцію `attack()` і матиме окрему реалізацію:

```

class Enemy {
public:
    virtual void attack() {
    }
};

class Ninja: public Enemy {
public:
    void attack() {
        cout << "Ninja!"<<endl;
    }
};

class Monster: public Enemy {
public:
    void attack() {
        cout << "Monster!"<<endl;
    }
};

```

```

int main() {
    Ninja n;
    Monster m;
    Enemy *e1 = &n;
    Enemy *e2 = &m;

    e1->attack();
    e2->attack();
}

/* Output:
Ninja!
Monster!
*/

```

Віртуальна функція є базовою функцією класу, яка оголошується з допомогою ключового слова **virtual**.

Зараз ми можемо використовувати покажчики **Enemy** для виклику функції `attack ()` .

```
e1->attack();
e2->attack();
```

Оскільки функція `attack()` оголошена віртуальною, вона працює як шаблон, повідомляючи про те, що похідний клас може мати власну функцію `attack ()` .

Наш ігровий приклад слугує для демонстрації концепції поліморфізму; ми використовуємо вказівники **Enemy** для виклику однієї і тієї ж функції **attack ()** та отримання різних результатів.

Якщо функція в базовому класі є **virtual**, реалізація функції у похідному класі викликається відповідно до фактичного типу об'єкта, на який посилається, незалежно від заявленого типу вказівника.

Клас, який оголошує або успадковує віртуальну функцію, називається **поліморфним класом**.

Віртуальні функції також можуть мати свою реалізацію в базовому класі:

```
class Enemy {
public:
    virtual void attack() {
        cout << "Enemy!"<<endl;
    }
};

class Ninja: public Enemy {
public:
    void attack() {
        cout << "Ninja!"<<endl;
    }
};

class Monster: public Enemy {
public:
    void attack() {
        cout << "Monster!"<<endl;
    }
};
```

```
int main() {
    Ninja n;
    Monster m;
    Enemy e;

    Enemy *e1 = &n;
    Enemy *e2 = &m;
    Enemy *e3 = &e;

    e1->attack();
    // Outputs "Ninja!"

    e2->attack();
    // Outputs "Monster!"

    e3->attack();
    // Outputs "Enemy!"
}
```

Віртуальні функції без визначення відомі як **чисто віртуальні функції**. Вони вказують, що похідні класи визначають цю функцію як власну.

```
class Enemy {
public:
    virtual void attack () = 0 ;
};
```

Чиста віртуальна функція в класі **Enemy** повинна бути перезаписана у похідних класах.

```
class Enemy {
public:
    virtual void attack() = 0;
};

class Ninja: public Enemy {
public:
    void attack() {
        cout << "Ninja!"<<endl;
    }
};

class Monster: public Enemy {
public:
    void attack() {
        cout << "Monster!"<<endl;
    }
};
```

Абстрактні класи

Ви **не можете** створити об'єкти базового класу з чисто віртуальною функцією. Запуск наступного коду поверне помилку:

```
Enemy e; // Error
```

Ці класи називають **абстрактними**. Це класи, які можуть бути використані лише як базові класи, і тому їм дозволяється мати чисті віртуальні функції.

З допомогою абстрактного базового класу можна створити покажчики та скористатись усіма своїми поліморфними здібностями.

Наприклад:

```
Ninja n;
Monster m;
Enemy *e1 = &n;
Enemy *e2 = &m;

e1->attack();
e2->attack();
```