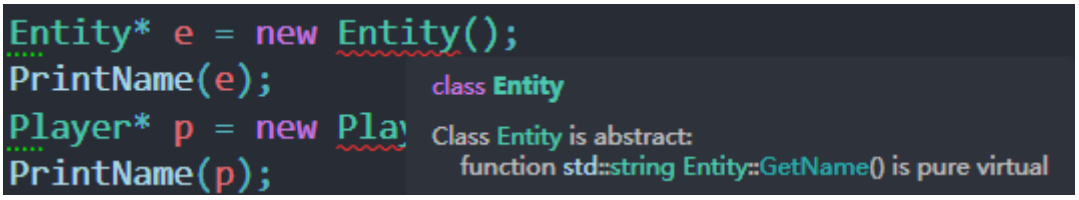
29 Interfaces in C++ (Pure Virtual Functions)

纯虚函数允许我们定义一个在基类中没有实现的函数，然后强制子类去实现这个函数

在OOP中，创建一个只包含未实现方法然后交由子类去实现功能的类是非常普遍的，这通常被称为*interface*(接口)，接口就是一个只包含未实现的方法并作为一个模板的类。
并且由于此接口类实际上不包含方法实现，所以我们无法实例化这个类。

```C++
class Entity
{
public:
    virtual std::string GetName() = 0;    // 仍被定义为虚函数，但是=0实际上将它变成了一个纯虚函数
};
```

意味着如果你想实例化这个类，那么这个函数必须在子类中实现。




```C++
class Player : public  Entity
{
private:
    std::string m_Name;
public:
    Player(const std::string& name)
        : m_Name(name) {}

    std::string GetName() override { return m_Name; } // 如果注释掉则不能实例化
};

void PrintName(Entity* entity)
{
    std::cout << entity->GetName() << std::endl;
}

int main()
{
    Entity* e = new Player("");  // 此时可以实例化
    PrintName(e);
    Player* p = new Player("Cherno");
    PrintName(p);
```

改成Player后就可以实例化了，仅仅是因为Player这个子类中实现了那个GetName函数。
所以很明显，你只能实例化一个实现了所有纯虚函数的类。

*Interface*就是C++中的类而已。

C++

```cpp
#include <iostream>
#include  <string>

class Printable
{
public:
    virtual  std::string GetClassName() = 0;
};

class Entity : public Printable
{
public:
    virtual std::string GetName() { return "Entity"; }
    std::string GetClassName() override { return "Entity"; }
};

class Player : public Entity
{
private:
    std::string m_Name;
public:
    Player(const std::string& name)
        : m_Name(name) {}

    std::string GetName() override { return m_Name; }
    std::string GetClassName() override { return "Player"; }
};

void PrintName(Entity* entity)
{
    std::cout << entity->GetName() << std::endl;
}


class A : public Printable
{
public:
    std::string GetClassName() override { return "A"; }
};

void Print(Printable* obj)
{
    std::cout << obj->GetClassName() << std::endl;
}

int main()
{
    Entity* e = new Entity();
    Player* p = new Player("Cherno");

    Print(e);
    Print(p);
    Print(new A());
```

```
    std::cin.get();
}
```

确保这个类有个特定的方法，那么就可以将这个类作为参数传入一个通用的函数，然后我们就可以调用这个函数或者做些其他事。