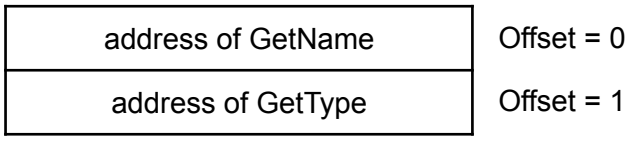
Let's say we start out with two classes:

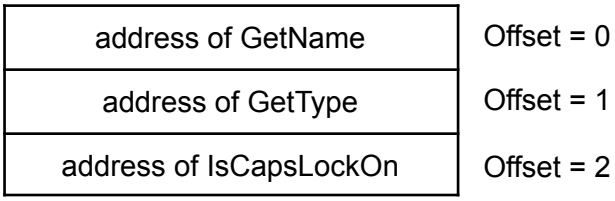
```
class IDevice {
public:
    virtual char const *GetName(void) = 0;
    virtual unsigned GetType(void) = 0;
};

class IKeyboard : public IDevice {
public:
    virtual bool IsCapsLockOn(void) = 0;
};
```

An object of type **IDevice** has a pointer at the beginning of it that points to the **IDevice-vtable**:`



An object of type **IKeyboard** has a pointer at the beginning of it that points to the **IKeyboard-vtable**:



The following code snippet:

```
int main(void)
{
    bool (IKeyboard::*p)(void) = &IKeyboard::IsCapsLockOn;
}
```

creates a member function pointer with the '*vtable offset*' set to **2**, because **IsCapsLockOn** is at offset 2.

SEE NEXT PAGE

If we were to add a new method to 'IDevice' as follows:

```
class IDevice {
public:
    virtual char const *GetName(void) = 0;
    virtual unsigned GetType(void) = 0;
    virtual bool IsEnabled(void) = 0; // new
};

class IKeyboard : public IDevice {
public:
    virtual bool IsCapsLockOn(void) = 0;
};
```

Then the two vtables would change as follows:

IDevice-vtable:`

address of GetName	Offset = 0
address of GetType	Offset = 1
address of IsEnabled	Offset = 2

IKeyboard-vtable:

address of GetName	Offset = 0
address of GetType	Offset = 1
address of IsEnabled	Offset = 2
address of IsCapsLockOn	Offset = 3

We have an ABI break here because the '*vtable offset*' of **IsCapsLockOn** has changed from **2** to **3**.

The solution to this problem is to mark the two classes as 'extensible' so that the compiler lays out the vtables differently.

SEE NEXT PAGE

So we mark the two classes as 'extensible' as follows:

```
class IDevice extensible {
public:
    virtual char const *GetName(void) = 0;
    virtual unsigned GetType(void) = 0;
};

class IKeyboard extensible : public IDevice {
public:
    virtual bool IsCapsLockOn(void) = 0;
};
```

The vtables are laid out differently now. An object of type **IDevice** has a pointer at the beginning of it that points to the vtable for **IDevice**:`

address of IDevice-methods-table-for-IDevice	Offset = 0

An object of type **IKeyboard** has a pointer at the beginning of it that points to the vtable for **IKeyboard**:

address of IDevice-methods-table-for-IKeyboard	Offset = 0
address of IKeyboard-methods-table-for-IKeyboard	Offset = 1

And so then we would have 3 more tables as follows:

IDevice-methods-table-for-IDevice:`

address of GetName	Offset = 0
address of GetType	Offset = 1

IDevice-methods-table-for-IKeyboard:

address of GetName	Offset = 0
address of GetType	Offset = 1

IKeyboard-methods-table-for-IKeyboard:

address of IsCapsLockOn	Offset = 0

In order to accommodate this new layout, *member function pointers* will become a little more complicated. Previously a *member function pointer* only had one '*vtable offset*', but now they will have two: a '***primary** vtable offset*' and a '***secondary** vtable offset*'. And so the following line of code:

```
bool (IKeyboard::*p)(void) = &IKeyboard::IsCapsLockOn;
```

creates a member function pointer with the '***primary** vtable offset*' set to **1,** and and the '***secondary** vtable offset*' set to **0**.

With this new vtable layout, it will be possible to add more virtual methods to **IDevice** without breaking the ABI for **IKeyboard**.