**virtual void funcFoo() const = 0;**

**VS**

**virtual void funcFoo() = 0;**

virtual void funcFoo() const = 0;

- You can't change the state of the object

- You can call this function via const objects

- You can only call another const member functions on this object

virtual void funcFoo() = 0;

- You can change the state of the object

- You can't call this function via const objects

virtual void func() const //in Base

virtual void func() //in Derived

const part is *actually a part* of the function signature, which means the derived class defines a *new*function rather than *overriding* the base class function. It is because their signatures don't match.

When you remove the const part, then their signature matches, and then compiler sees the derived class definition of func as *overridden version* of the base class function func, hence the derived class function is called if the runtime type of the object is Derived type. This behavior is called runtime polymorphism.

virtual const int getNumber();

const virtual int getNumber();

As was already said, there is no difference. However, note that these two **do** differ:

virtual const int getNumber();

virtual int getNumber() const;

In the first method, const refers to the returned value of type int.

In the second method, const refers to the object the method is called on; that is, this will have type T const \* inside this method, - you will be able to call only const methods, modify only mutable fields and so on.