27|10|2023

Object Oriented Programming Class 03

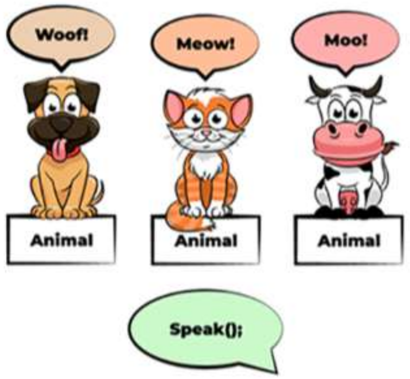
1. Run Time Polymorphism

What is run time polymorphism?
It is a late binding and dynamic polymorphism.

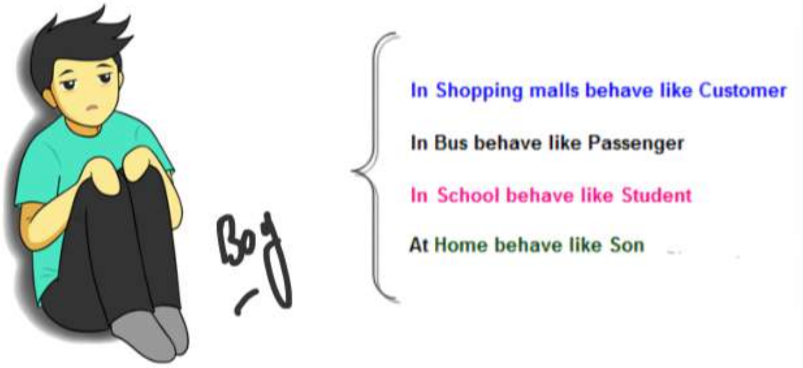
late binding or dynamic polymorphism is a binding mechanism that occurs during runtime.

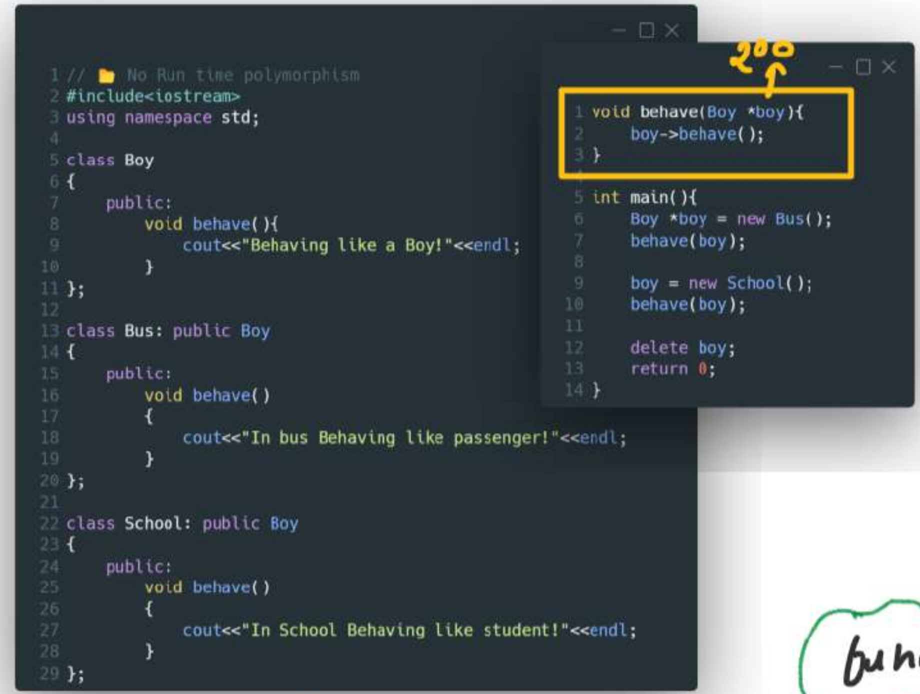
This means that the method that is called is determined when the program is running, not when it is compiled.

Ex



Ex


Boy
In Shopping malls behave like Customer
In Bus behave like Passenger
In School behave like Student
At Home behave like Son



```cpp
// 📁 No Run time polymorphism
#include<iostream>
using namespace std;

class Boy
{
    public:
        void behave(){
            cout<<"Behaving like a Boy!"<<endl;
        }
};

class Bus: public Boy
{
    public:
        void behave()
        {
            cout<<"In bus Behaving like passenger!"<<endl;
        }
};

class School: public Boy
{
    public:
        void behave()
        {
            cout<<"In School Behaving like student!"<<endl;
        }
};
```

200

```cpp
void behave(Boy *boy){
    boy->behave();
}

int main(){
    Boy *boy = new Bus();
    behave(boy);

    boy = new School();
    behave(boy);

    delete boy;
    return 0;
}
```

```
OUTPUT:
Behaving like a Boy!
Behaving like a Boy!
```

Yeh Output Kyu Aa rha he.?

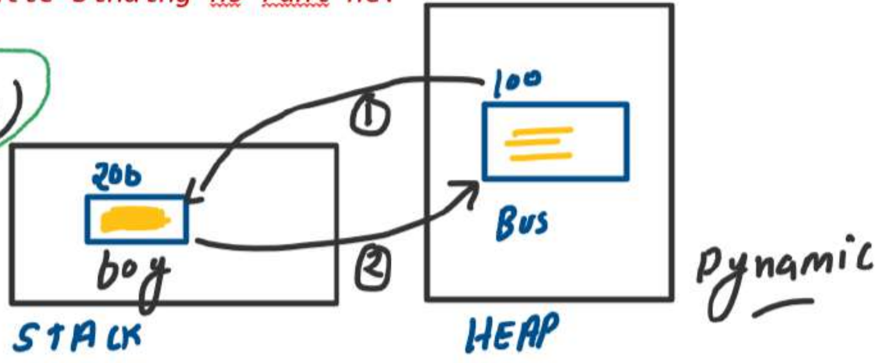
*Boy *boy = new Bus();*

Problem yeh he ki compile time par early/static binding ho rahi he.

behave (200)

(200 Add pass kiya ja Rha vai)

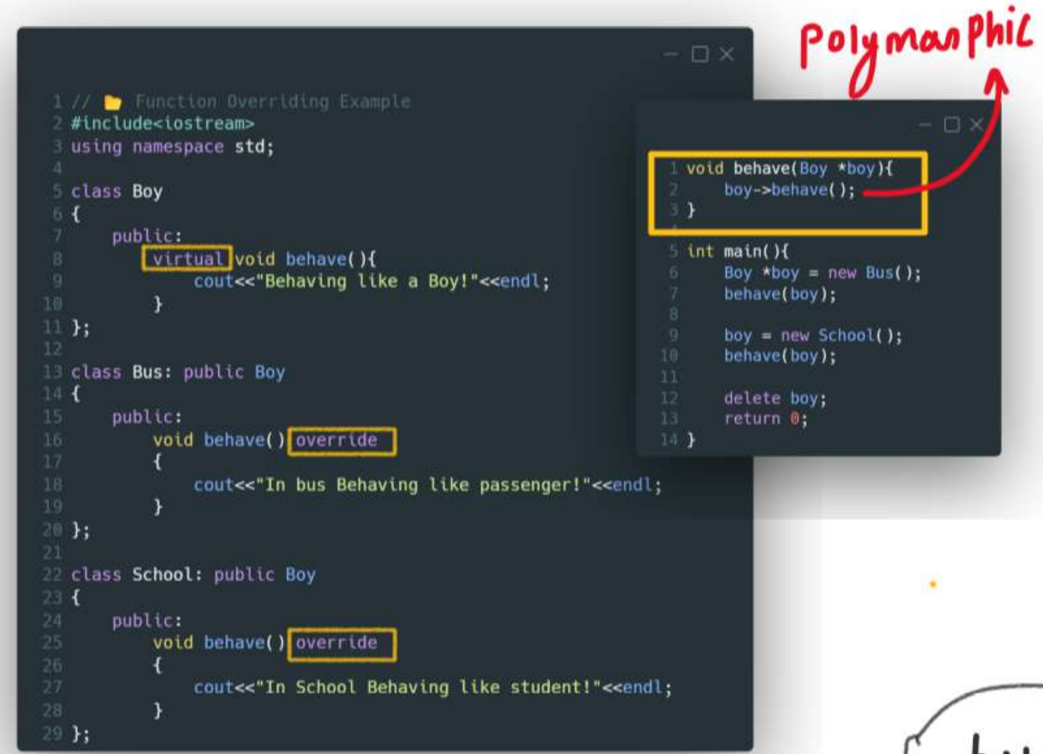
static



Dynamic

Ans Late/Dynamic Binding

Solution Kya Hi? Ki 100 pass kar Pau.-



```cpp
// 📁 Function Overriding Example
#include<iostream>
using namespace std;

class Boy
{
    public:
        virtual void behave(){
            cout<<"Behaving like a Boy!"<<endl;
        }
};

class Bus: public Boy
{
    public:
        void behave() override
        {
            cout<<"In bus Behaving like passenger!"<<endl;
        }
};

class School: public Boy
{
    public:
        void behave() override
        {
            cout<<"In School Behaving like student!"<<endl;
        }
};
```

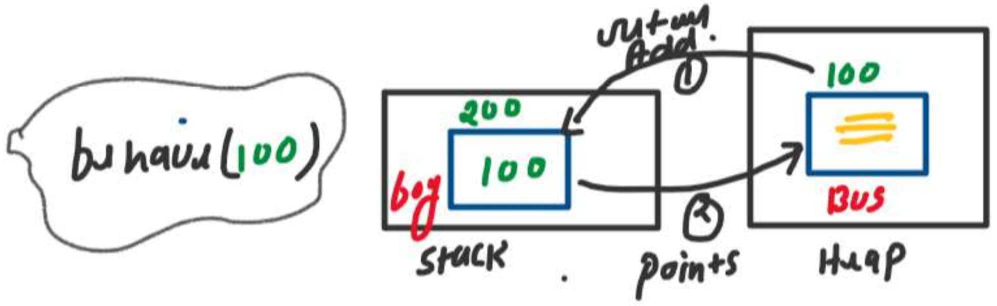
```cpp
void behave(Boy *boy){
    boy->behave();
}
int main(){
    Boy *boy = new Bus();
    behave(boy);

    boy = new School();
    behave(boy);

    delete boy;
    return 0;
}
```

```
OUTPUT:
In bus Behaving like passenger!
In School Behaving like student!
```

Yeh Output Kyu Aa rha he.?

*Boy *boy = new Bus();* ← upcasting

Jab function virtual hota hai to compile time par us function ko seriously nhi liya jata hai. Jabki use Run time par decision lena hota hai ki kis object ke function ko call karna hai.



Upcasting Concept:

```
Parent *p = new Child();
```

Upcasting is using the Super/Parent class's reference or pointer to refer to a Sub/Child class's object.

Down casting Concept:

```
Child *c = new Parent();
```

Down casting is using the Sub/Child class's reference or pointer to refer to a Super/Parent class's object.

Function Overriding:

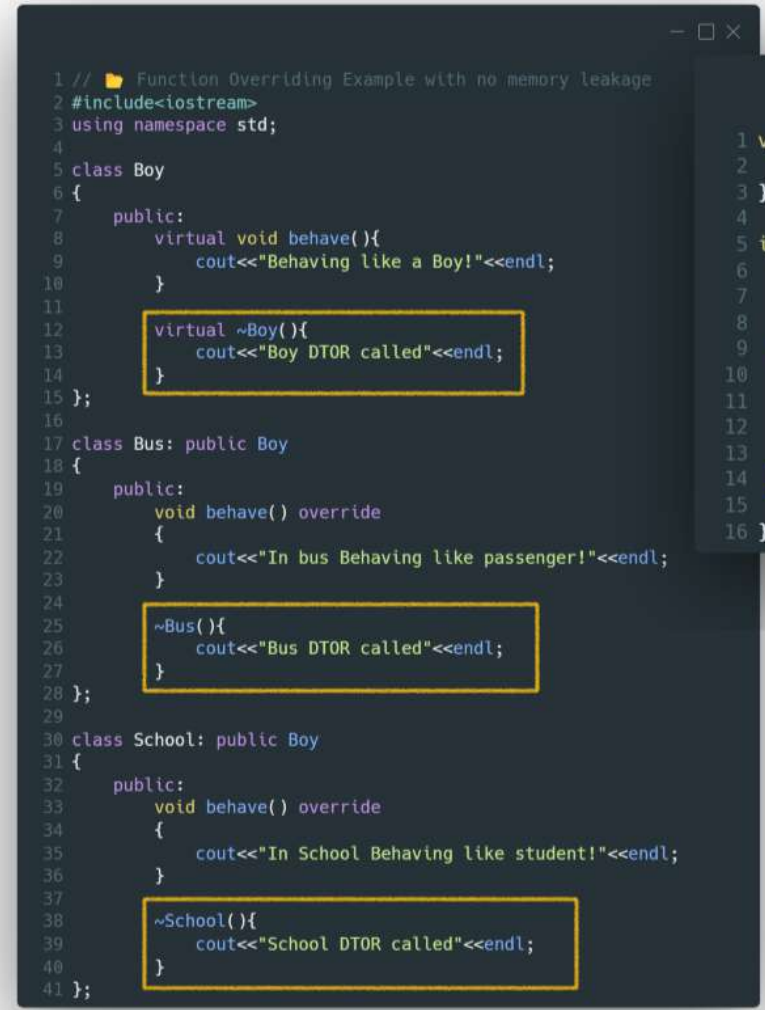
The process of defining a function with the same name and parameters as a function in a base class, in a derived class.
In short, A feature that allows us to use a function in the child class that is already present in its parent class.

What is virtual keyword:

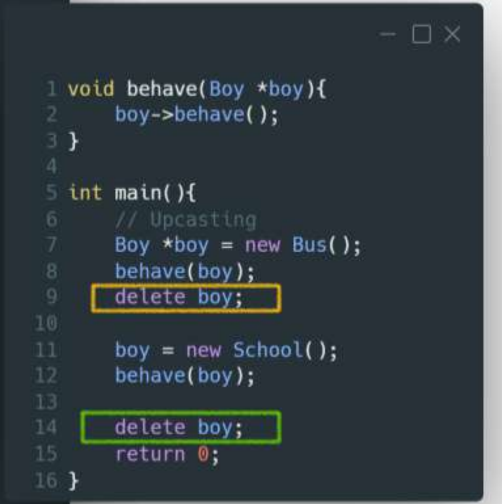
1. *Yeh ek late binding karne ka tarika hai*
2. *Yeh ek run time par decision lene tarika hai*
3. *Yeh ek tarika hai function ko virtual bnane par usko compile time par seriously na le ka*

Note Points:

1. *Virtual function can always be written in parent class.*
2. *Without virtual early binding, static binding*
3. *With virtual late binding, dynamic binding*



```cpp
// 📁 Function Overriding Example with no memory leakage
#include<iostream>
using namespace std;

class Boy
{
    public:
        virtual void behave(){
            cout<<"Behaving like a Boy!"<<endl;
        }

        virtual ~Boy(){
            cout<<"Boy DTOR called"<<endl;
        }
};

class Bus: public Boy
{
    public:
        void behave() override
        {
            cout<<"In bus Behaving like passenger!"<<endl;
        }

        ~Bus(){
            cout<<"Bus DTOR called"<<endl;
        }
};

class School: public Boy
{
    public:
        void behave() override
        {
            cout<<"In School Behaving like student!"<<endl;
        }

        ~School(){
            cout<<"School DTOR called"<<endl;
        }
};
```



```cpp
void behave(Boy *boy){
    boy->behave();
}

int main(){
    // Upcasting
    Boy *boy = new Bus();
    behave(boy);
    delete boy;

    boy = new School();
    behave(boy);

    delete boy;
    return 0;
}
```

OUTPUT:

```
✓ In bus Behaving like passenger!
1 Bus DTOR called
2 Boy DTOR called
✓ In School Behaving like student!
3 School DTOR called
4 Boy DTOR called
```