Data Structures and Object Oriented Programming

Lecture 19

Dr. Naveed Anwar Bhatti

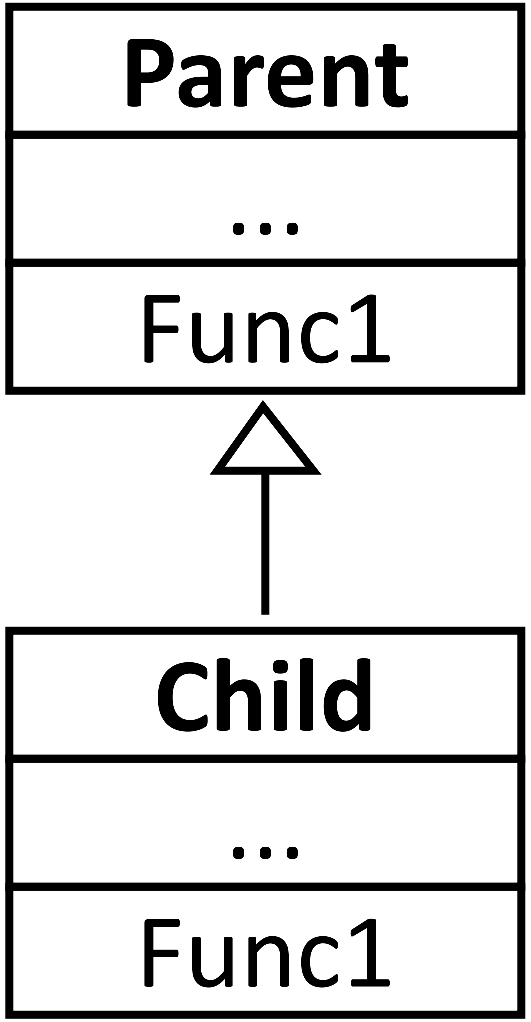
Webpage: naveedanwarbhatti.github.io

Overriding vs. Overloading

Overriding

- Derived class can override (modify) the member functions of its base class

- To override a function the derived class simply provides a function with the same signature as that of its base class



Overriding (example)

```cpp
#include <iostream>
using namespace std;

class Parent{
public:
    void myFunction()
    {
        cout << “Parent class“ << endl;
   }
};
```

```cpp
class Child : public Parent {

public:
    void myFunction()
    {
        cout << “Child class“ << endl;
    }

};


int main()
{
    Child myObj;
    myObj.myFunction();
    return 0;
}
```

Overloading vs. Overriding

- Overloading is done within the scope of one class

- Overriding is done in scope of parent and child

- Overriding within the scope of single class is error due to duplicate declaration

- <u>Example</u>

```
class Parent{
public:
    void myFunction()
    {
        cout << “Hello 1“ << endl;
    }
    void myFunction()
    {
        cout << “Hello 2“ << endl;
    }

};
```

Error

Overriding Member Functions of Base Class

- Derive class can override member function of base class such that the working of function is **totally changed**

- Derive class can override member function of base class such that the working of function **is based on former implementation**

Overriding Example (example)

```
#include <iostream>
using namespace std;

class Parent{
public:
    void myFunction()
    {
        cout << “Parent class“ << endl;
    }
};
```

```
class Child : public Parent {

public:
    void myFunction()
    {
        myFunction();
       cout << “Child class“ << endl;
     }

};


int main()
{
    Child myObj;
    myObj.myFunction();
    return 0;
}
```

Code will stuck in recursive call

We use scope operator

```
#include <iostream>
using namespace std;

class Parent{
public:
    void myFunction()
    {
        cout << “Parent class“ << endl;
    }
};
```

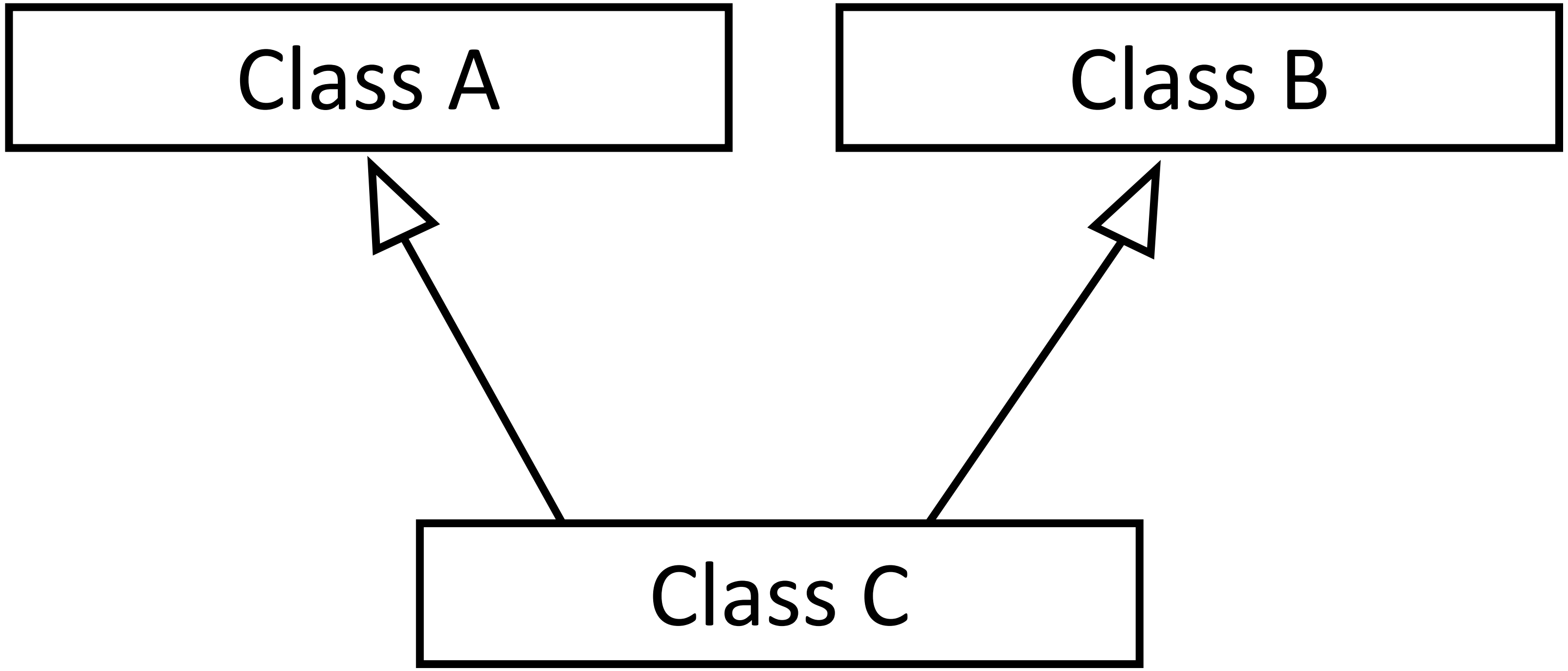
```
class Child : public Parent {

public:
    void myFunction()
    {
        Parent::myFunction();
        cout << “Child class“ << endl;
     }

};


int main()
{
    Child myObj;
    myObj.myFunction();
    return 0;
}
```

Another Example



Class A

Class B

Class C

Another Example

```
class A {
public:
    void myFunction() {
    cout << “class A“ << endl;
   }
};

class B : {
public:
    void myFunction() {
    cout << “class B" << endl;
    }
};
```

```
class C : public A, public B
{

};


int main()
{
    C myObj;
    myObj.myFunction();
    return 0;
}
```

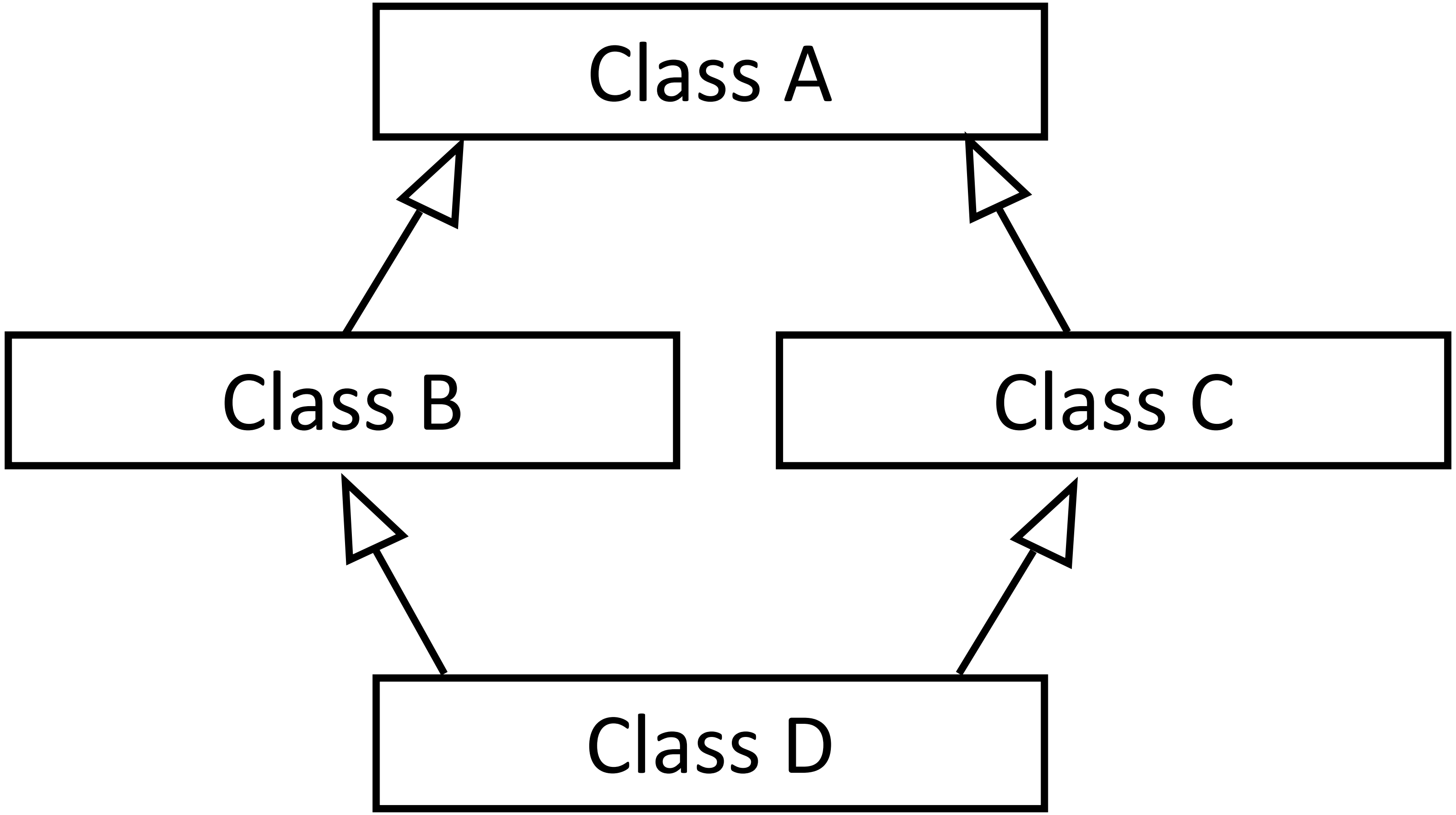
Error

myObj.A::myFunction();

Or

myObj.B::myFunction();

Another Scenario (Hybrid Inheritance)



Diamond Problem

```
class A {
public:
    void myFunction() {
    cout << “class A“ << endl;
   }
};

class B : public A{
public:
    void myFunction() {
    cout << “class B" << endl;
    }
};

class C : public A{
public:
    void myFunction() {
    cout << “class C" << endl;
    }
};
```

```
class D : public B, public C{
public:
    void myFunction() {
    cout << “class D" << endl;
    }
};



int main() {
    D myObj;
    myObj.myFunction();
    myObj.B::myFunction();
    myObj.A::myFunction();

   return 0;
}
```

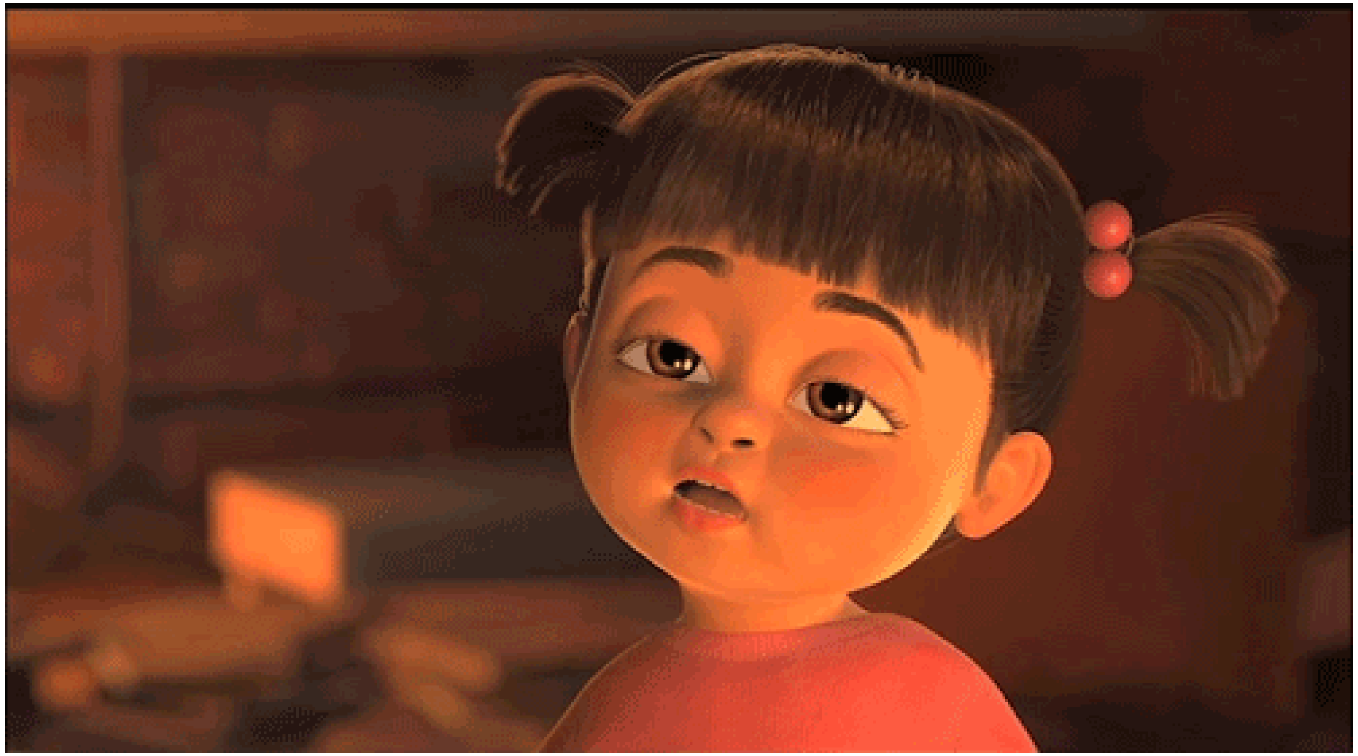
Error

myObj.B::A::myFunction();

Or

myObj.C::A::myFunction();

Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over