# IT206 Data Structures Lab with OOP

## Lecture 3

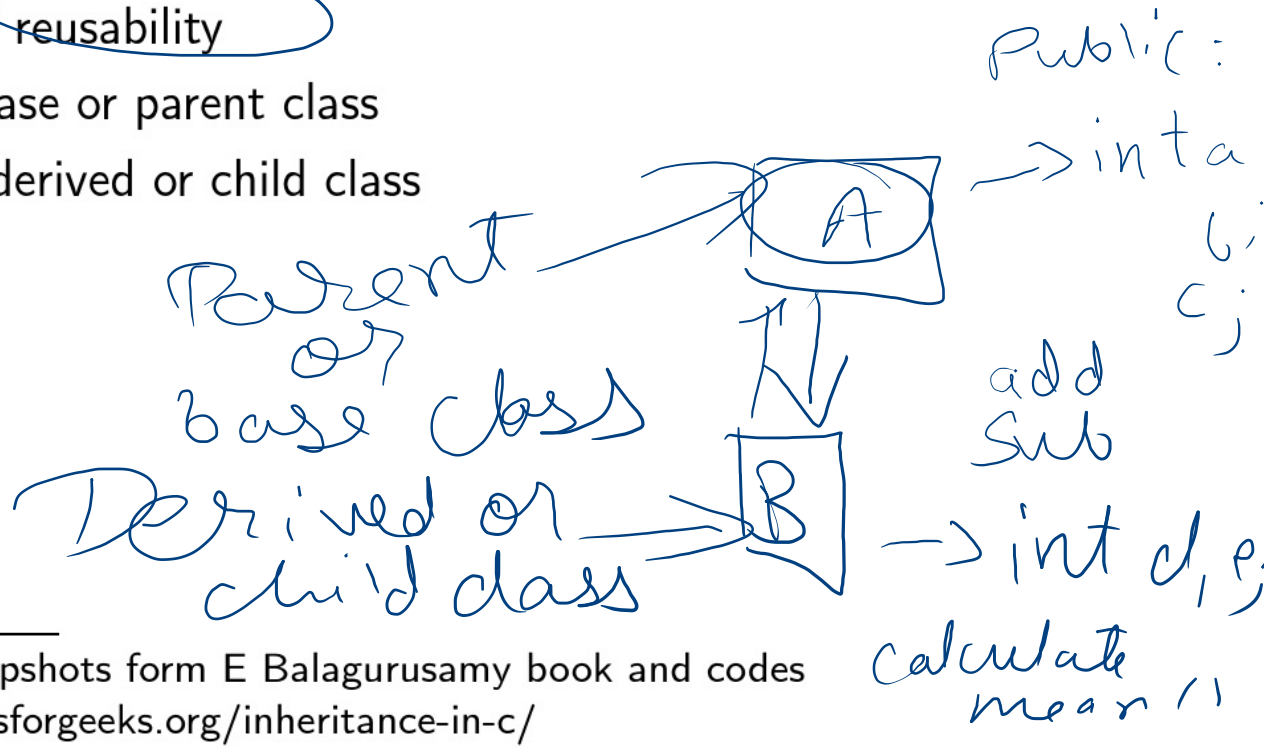
**Rachit Chhaya**

DA-IICT

April 8, 2022

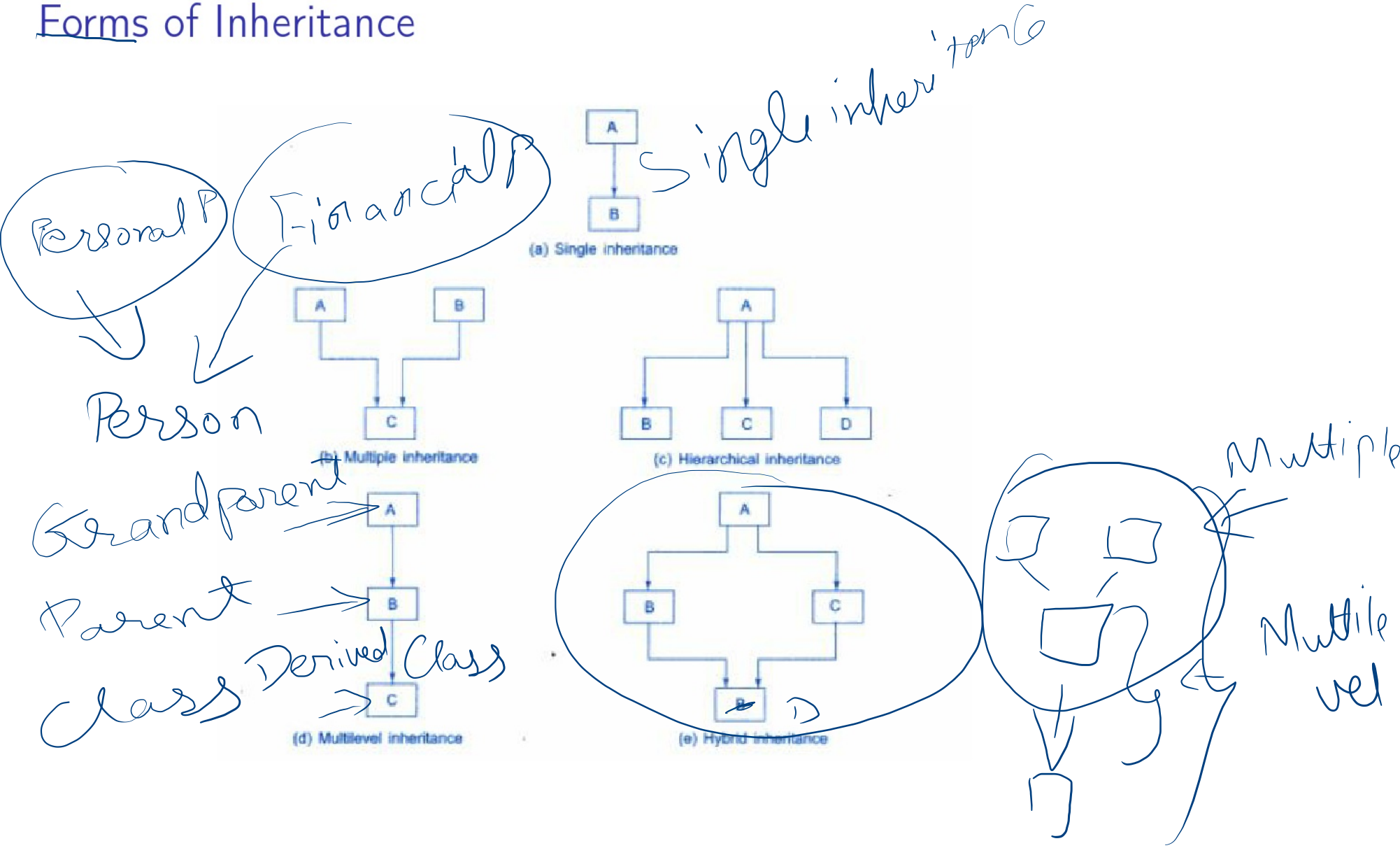
# Inheritance[1]

- Mechanism of deriving a new class from an old one
- Supports the concept of reusability
- The old class is called base or parent class
- The new class is called derived or child class




[1]Figures in these slides are snapshots form E Balagurusamy book and codes are taken from https://www.geeksforgeeks.org/inheritance-in-c/

# Forms of Inheritance



class Table : public Furniture

```
class derived-class-name  : visibility-mode base-class-name
{
        .....//
        .....//   members of derived class
        .....//
};
```

Private

Examples:

Privately derived

```
class ABC: private XYZ        // private derivation
{
        members of ABC
};

class ABC: public XYZ         // public derivation
{
        members of ABC
};

class ABC: XYZ                // private derivation by default
{
        members of ABC
};
```

Private

- **Private Inheritance**: Public members of Base Class become Private members of Derived Class

- **Private Inheritance**: Public members of Base Class become Private members of Derived Class

- **Public Inheritance**: Public members of Base Class become Public members of derived class

B. b = 4;

- **Private Inheritance**: Public members of Base Class become Private members of Derived Class
- **Public Inheritance**: Public members of Base Class become Public members of derived class
- Private Members of Class are not inherited.

class A
{ functions
}

class B
{
function
}

- **Private Inheritance**: Public members of Base Class become Private members of Derived Class

- **Public Inheritance**: Public members of Base Class become Public members of derived class

- Private Members of Class are not inherited.

Classname :: A. fun1

- What about functions having same name in both private and public class?

B b

b.fun1()

b.fun1()
b.fun2()

Class A
{
Public
fun2();
}

class B : public
{
Public
fun1();

# Protected Variables

- What if want a class to inherit private variables??

# Protected Variables

- What if want a class to inherit private variables??
- Solution: Protected Variables!!

# Protected Variables

- What if want a class to inherit private variables??
- Solution: Protected Variables!!

Private characteristic

- Protected member accessible by member functions within its class and any class immediately derived from it.

some sort of public

# Protected Variables

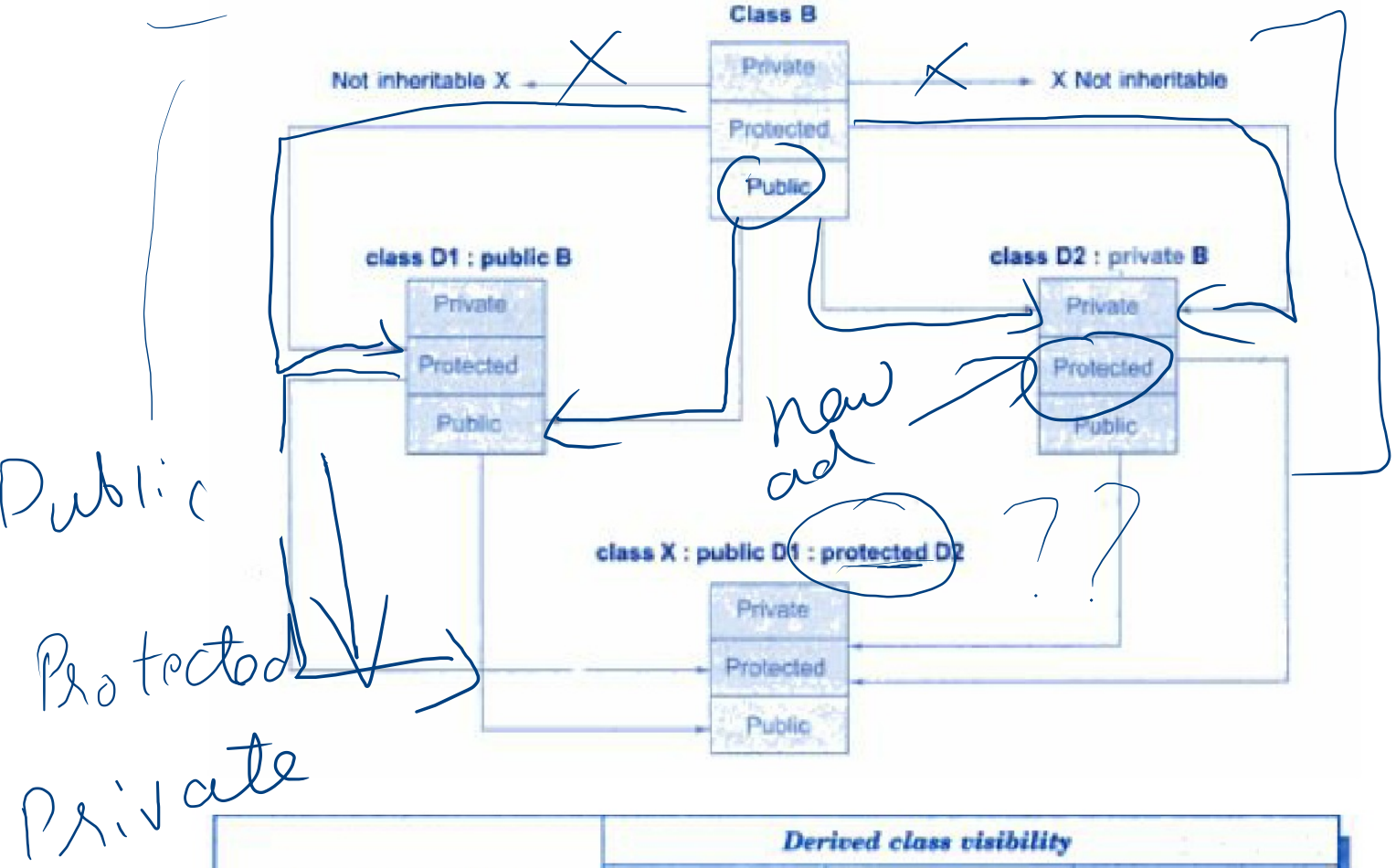
- What if want a class to inherit private variables??
- Solution: Protected Variables!!

- Protected member accessible by member functions within its class and any class immediately derived from it.

- Protected member inherited in public mode - Protected in derived class
- Protect member inherited in Private mode- Private in derived class

Class A
{
protected:
int A;
B;
}

Private X




| Base class visibility | | Derived class visibility | | |
|---|---|---|---|---|
| | | ***Public derivation*** | ***Private derivation*** | ***Protected derivation*** |
| Private | ⟶ | Not inherited | Not inherited | Not inherited |
| Protected | ⟶ | Protected | Private | Protected |
| Public | ⟶ | Public | Private | Protected |

# Single Public Inheritance

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

// base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};

// sub class derived from a single base classes
class Car : public Vehicle {

};
```

Car.fun

Derived

```
// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```

A : : A(

# Multiple Inheritance

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

// first base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};

// second base class
class FourWheeler {
  public:
    FourWheeler()
    {
      cout << "This is a 4 wheeler Vehicle\n";
```

```
// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {

};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

Derived class

# Multilevel Inheritance

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

// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};

// first sub_class derived from class vehicle
class fourWheeler: public Vehicle
{  public:
    fourWheeler()
    {
```

cout << "This is a wheeler\n";

}

};

```
// sub class derived from the derived base class fourWheele
class Car: public fourWheeler {
   public:
     Car()
     {
       cout << "Car has 4 Wheels\n";
     }
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

? ?

# Hierarchical Inheritance

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

// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};
```

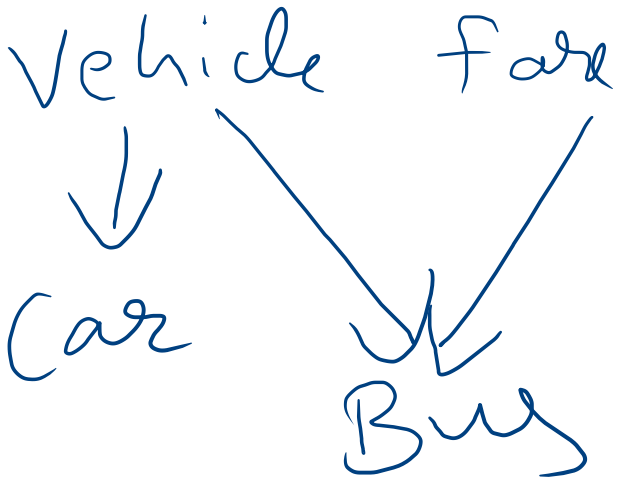
```
// first sub class
class Car: public Vehicle
{

};
// second sub class
class Bus: public Vehicle
{

};
// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Car obj1;
    Bus obj2;
    return 0;
}
```

# Hybrid Inheritance

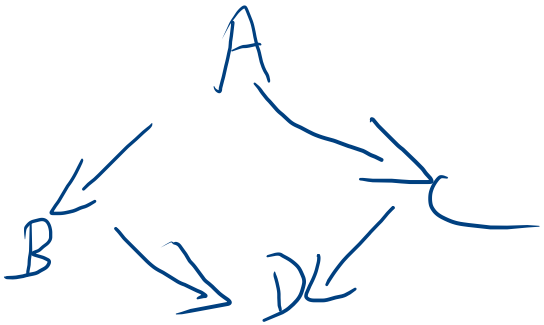
```
#include <iostream>
using namespace std;

// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle\n";
    }
};

//base class
class Fare
{
    public:
    Fare()
```

{ cout << "Fare is costly" ; }

```
// first sub class
class Car : public Vehicle
{

};
// second sub class
class Bus : public Vehicle, public Fare
{

};
// main function
int main()
{
    Car obj1;
    Bus obj2;
    return 0;
}
```



# Hybrid Inheritance : Multipath




```
#include <iostream>
class ClassA {
  public:
    int a;
};

class ClassB : public ClassA {
  public:
    int b;
};

class ClassC : public ClassA {
  public:
    int c;
};

class ClassD : public ClassB, public ClassC {
  public:
```

class ClassB
{
public
int a,b;
}

int d;

```
int main()
{
    ClassD obj;
    // obj.a = 10;                    // Statement 1, Error
    // obj.a = 100;                   // Statement 2, Error
    obj.ClassB::a = 10;  // Statement 3
    obj.ClassC::a = 100; // Statement 4
    obj.b = 20;
    obj.c = 30;
    obj.d = 40;

    cout << " a from ClassB  : " << obj.ClassB::a;
    cout << "\n a from ClassC  : " << obj.ClassC::a;

    cout << "\n b : " << obj.b;
    cout << "\n c : " << obj.c;
    cout << "\n d : " << obj.d << '\n';
    return 0;
}
```

obj.a = 10;
obj.a = 100;