

Concepts on OOP

Inheritance

It's a method to create class from existing class with all properties and characteristics and the ability to extend and modify

Class vehicle -> Fuel amount

-> Display fuel

-> Apply break

Vehicle properties is common between:

|> Class truck

|> Class Car

|> Class Bus

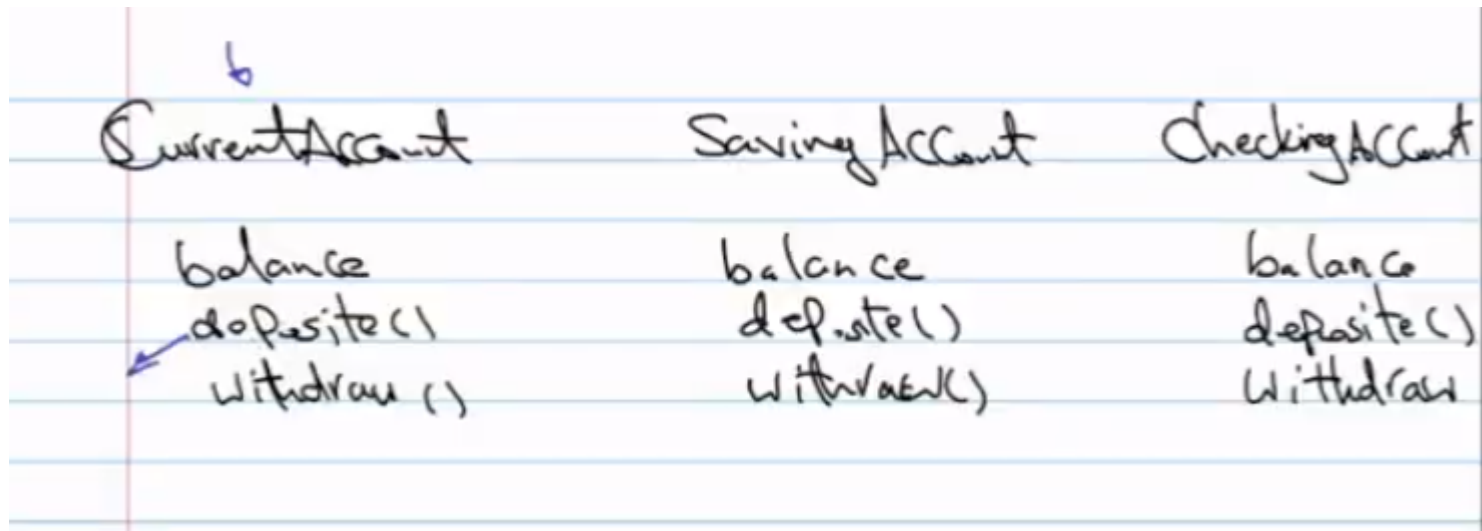
- **Ability to extend**

Old class New class

(Members) (Old class + New features)

- **Ability to modify**

Example:



CurrentAccount	SavingAccount	CheckingAccount
balance	balance	balance
deposit()	deposit()	deposit()
withdraw()	withdraw()	withdraw()

deposit implementation differ between each class, so we implement general class and re-implement this method in each class

Terminologies

Base Class (Super or Parent class): class from which we inherit another class

Derived Class (Child class): Class inherit base class

```
class A
{
    int x;
    public:
        void display();
}
/*A is the parent class, B is the child class which have x and y and 2 display methods */
class B: public A
{
    int y;
    public:
        void display_y();
}
```

Access modifiers

Protected is the same as private if we have only base class, but we can access it in child class

Protected member can be accessed in its class and any class inherited the base class

```
class A{
    private:
        int y;
    protected:
        int x;
};

class B: public A{
    public:
        void setX(int i){
            x = i; // x is protected member of class A and can be accessed in class B
        }
        void setY(int i){
            y = i; // ERROR y is private member of class A and cannot be accessed in class B
        }
};
```

Note: Default access modifier is private

```
class A{
    //Default access specifier is private
    int z;
};

class B: public A{
    public:
        void setZ(int i){
```

```
        z = i; // ERROR, z is private member of class A and cannot be accessed in class B
    }
};
```

Same for class itself

```
//C inherits from A, but A is a private base class of C as default access specifier is private
class C: A{

};
```

Note:

```
//Protected inheritance
//Public and protected members of the base class become protected members of the derived class
//Private members of the base class are not accessible in the derived class
class D: protected A{

};
```

Class A

{ int x

protected:

int y

public:

int z

Public

→

Class B: public A

{

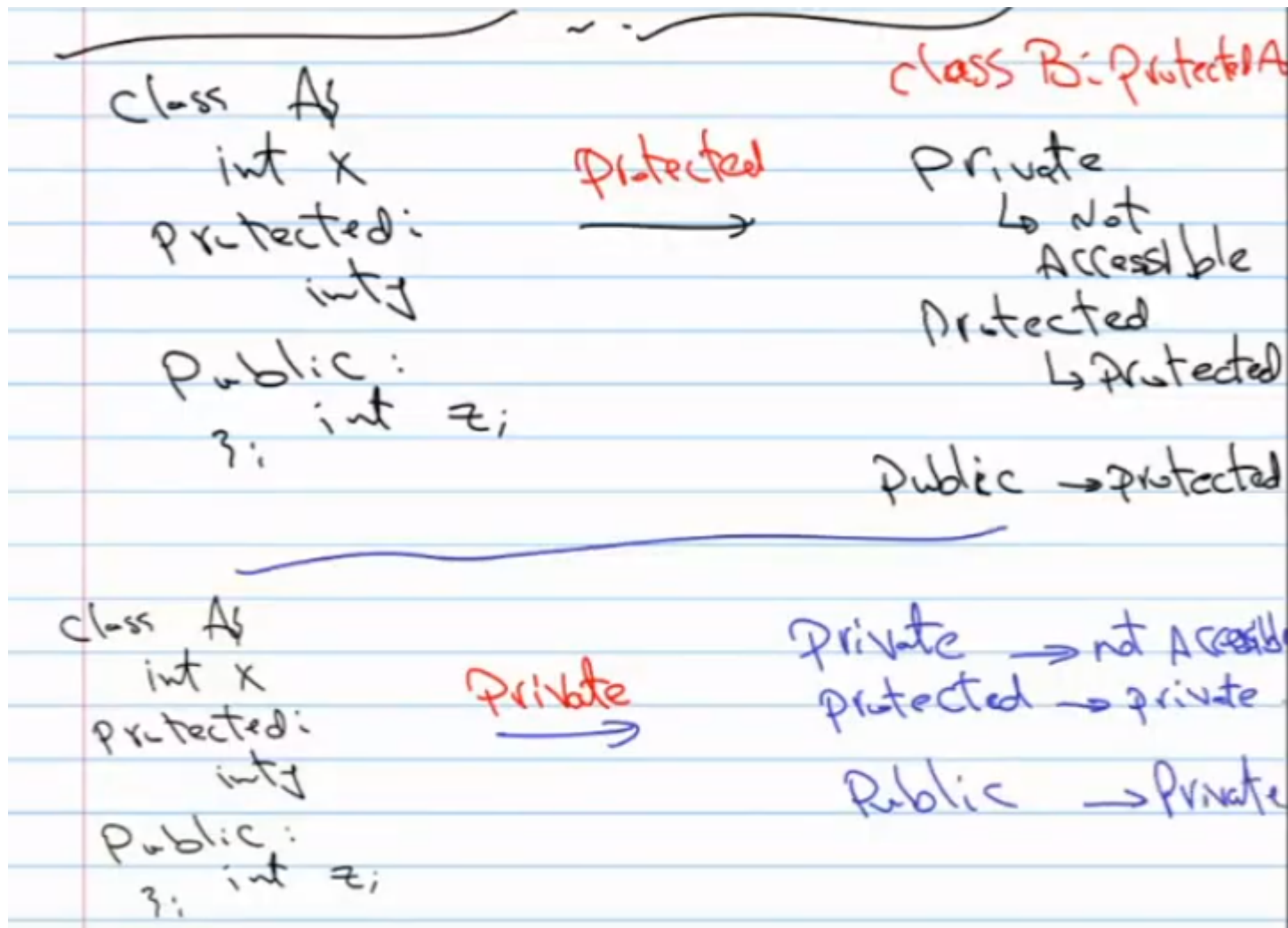
}

protected → protected

public → public

private → Not

Accessible



CONCLUSION

```
class A{  
    //Default access specifier is private  
    int z;  
    private:  
        int y;
```

```

    protected:
        int x;
    public:
        int w;
};

class B: public A{
    public:
        void setX(int i){
            x = i; // x is protected member of class A and can be accessed in class B
        }
        /**
         * setY and setZ are not accessible in class B as they are private members of class A
         */
        void setW(int i){
            w = i; // w is public member of class A and can be accessed in class B
        }
};

//C inherits from A, but A is a private base class of C as default access specifier is private
class C: A{
    public:
        void setX(int i){
            x = i; // ERROR, x is protected member of class A and cannot be accessed in class C
        }
        /**
         * setY and setZ are not accessible in class C as they are private members of class A
         */
        void setW(int i){
            w = i; // w is public member of class A and can be accessed in class C
        }
};

//Protected inheritance

```

```

//Public and protected members of the base class become protected members of the derived class
//Private members of the base class are not accessible in the derived class
class D: protected A{
    public:
        void setX(int i){
            x = i; // x is protected member of class A and can be accessed in class D
        }
        /**
         * setY and setZ are not accessible in class D as they are private members of class A
         */
        void setW(int i){
            w = i; // w is public member of class A and can be accessed in class D
        }
};

int main(){
    //B inherits publicly from A so public and protected members of A are accessible in B
    //but protected cannot be accessed in main
    B b;
    b.x = 10; // ERROR x is protected member of class A and it protected in class B
    b.w = 40; // w is public member of class A and can be accessed in class B

    //C inherits privately from A so public and protected members of A are private in C
    C c;
    //c.x = 10; // ERROR x is protected member of class A and it is private in class C
    //c.y = 20; // ERROR y is private member of class A and it is private in class C
    //c.w = 40; // ERROR w is public member of class A and it is private in class C

    //D inherits protectedly from A so public and protected members of A are accessible in D
    D d;
    d.x = 10; // ERROR x is protected member of class A and it protected in class D
    d.y = 20; // ERROR y is private member of class A and it is not accessible in class D
    d.w = 40; // ERROR w is public member of class A and protected in class D
}

```



```

    return 0;
}

```

Member Access Inheritance Type	Public	Protected	Private
Public	Public	Protected	Not Accessible
Protected	Protected	Protected	~
Private	Private	Private	~

Calling constructor and Destructor

```

/**
 * Which constructor and destructor is called first in inheritance?
 */
#include <iostream>
using namespace std;

class Base{
    int x;
public:
    Base(): x(10){

```

```

        cout << "Base Constructor" << endl;
    }
    ~Base(){
        cout << "Base Destructor" << endl;
    }
    void setX(int i){
        x = i;
    }
    int getX(){
        return x;
    }
};

class Derived: public Base{
    int y;
public:
    Derived(): y(20){
        cout << "Derived Constructor" << endl;
    }
    ~Derived(){
        cout << "Derived Destructor" << endl;
    }
    void setY(int i){
        y = i;
    }
    int getY(){
        return y;
    }
};

int main(){
    //Base class constructor is called first and then derived class constructor is called
    Derived d;
    d.setX(5);
    d.setY(15);
    cout << "X: " << d.getX() << endl;
}

```

```
    cout << "Y: " << d.getY() << endl;
    return 0;
}
```

Output:

```
• safia@safia:~/CPP/Learning-CPP/SessionLabs/Session5$ ./Lab2
Base Constructor
Derived Constructor
X: 5
Y: 15
Derived Destructor
Base Destructor
```

Note:

Add Parameterized constructor for each class then:

```
int main(){

    Derived b{5}; //Default constructor of base class is called not parameterized constructor
```

but if we need to call parameterized constructor of base when we create parameterized object of derived class we can invoke it like this:

```
    Derived(int y): Base(y), y{y}{
        cout << "Derived Constructor Derived(int y):y(y)" << endl;
    }
```

Then the output will be

```
safia@safia:~/CPP/Learning-CPP/SessionLabs/Session5$ ./Lab2
Base Constructor Base(int x):x(x)
Derived Constructor Derived(int y):y(y)
```

Derived Destructor

Base Destructor

Note:

We can not modify base constructor and destructor or overloading operators or friend function in the derived class