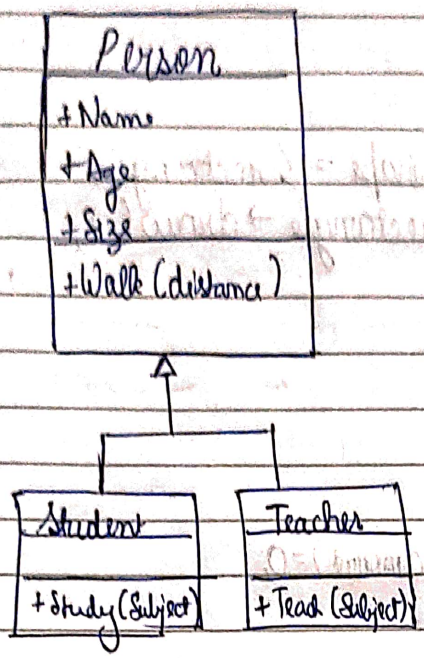
Low Level Design

Lecture # 1

① Inheritance - IS a relationship



Constructor

```
Student (std::string n, float g) : name(n), grade(g) { }
Student student ("John", 85.5);
```

Example

→ abstract base class

```
class Shape
{
public:
    virtual void draw() = 0;
};

class Circle : public Shape
{
public:
    virtual void draw override
    {
        std::cout << "Draw circle" << std::endl;
    }
};
```

```
int main()
{
    Shape *shape;      → pointer can point to anything inheriting from shape
    Circle circle;
    Rectangle rectangle;

    shape = &circle;          | shape = &rectangle;
    shape->draw();            | rectangle->draw();
}
```

Example

```
class Payment {
public:
    virtual void pay(float amount) = 0;
};

class CreditCard: public Payment
{
public:
    void pay(float amount) override
    {
        cout << "Paid" << amount << "using CC" << endl;
    }
};

class DebitCard: public Payment { -- };

void executePayment(Payment *paymentMethod, float amount)
{
    paymentMethod->Pay(amount);
}
```

```
int main()
{
    Payment *payment;
    CreditCard cc;
    DebitCard dc;

    payment = &cc;
    executePayment(payment, 20.0);
    payment = &dc;
    executePayment(payment, 50.0);
}
```

Q.
```
#include <iostream>
struct A {
    void g() { std::cout "A::g"; }
    virtual void f() { std::cout "A::f"; }
};
struct B: A {
    void g() { cout << "B::g"; }
    void f() override { std::cout "B::f"; }
};
int main() {
    A* p = new B();
    p->g();
    p->f();
}
```

A::g
B::f

} since g() is not virtual, C++ uses static binding, meaning compiler looks at the type of the pointer, not the actual object it points too