FACTORY DESIGN PATTERN

A simple use case

What goes on in a factory?

- As a customer, you go to a shoe making factory and request for a particular type of shoe
- Probably for school, party, or church
- The factory then asks you to wait at the entrance, or a waiting room for customers.
- Whatever goes on in the manufacturing room, you see them not
- The direct communication interface is the customer
- Every shoe type machine have common interfaces to how a general shoe is made.

Maintainability advantage

- Primarily allowing sub classes to choose the type of object to create
- Loose coupling which allows components function independently

```
/**
 * A shoe as our product has the folowing
 * show year of manufactur
 * color
 * size
*/

enum shoeType
{
    church,
    school,
};
```

```
class Shoe
{
public:
    size_t size;
    std::string color;
    int year;
    void setinfo(shoeType type)
    {
        if (type == church)
        {
            size = 34;
            year = 1934;
            color =  "brown";
        }
        else if (type == school)
        {
            size = 37;
            year = 1937;
            color = "white";
        }
    }
    virtual void printinfo() = 0;
    ~Shoe() {}
};
```

```cpp
// two shoes
class shoeChurch : public Shoe
{
    void printinfo() override
    {
        //setinfo(type);
        std::cout << "shoe size is " << size << std::endl
                  << "shoe color is " << color << std::endl
                  << "shoe year is " << year;
    }
};

class shoeSchool : public Shoe
{
    void printinfo() override
    {
        //setinfo(type);
        std::cout << "shoe size is " << size << std::endl
                  << "shoe color is " << color << std::endl
                  << "shoe year is " << year;
    }
};
```

```
class shoeMaker
{
public:
    Shoe* makeShoeType(shoeType shoe)
    {
        if (shoe == church)
        {
            return new shoeChurch();
        }
        else if (shoe == school)
        {
            return new shoeSchool();
        }
    }
};
```

```
/**
 * logically, a customer has no shoe
 * after he requests before he can get
*/
class customer
{
public:
    // the customer instructs the manufacturer to work
    customer(shoeType type)
    {
        shoeMaker* shoetype = new shoeMaker();
        shoe = shoetype->makeShoeType(type);
        shoe->setinfo(type);
        delete shoetype;
    }
    // request for shoe.. if he has not yet, he gets null
    Shoe* getshoe()
    {
        return shoe;
    }
private:
    // the customer's shoe.. either null or a shoe
    Shoe* shoe = nullptr;
};
```

```
int main()
{
    customer* shoe =  new customer(church);
    shoe->getshoe()->printinfo();
}
```

Direct customer interface