

Philip Pesic

Week 17

December 11 2022

Week 17 Program

```
// Factory Pattern
```

```
// * Add code for a 4th pizza type: Cheese
```

```
// * Modify the pizza_information function, so it also prints the pizza name
```

```
// * Comment each function - Explaining what it does
```

```
// * Explain why this is a factory pattern
```

```
//
```

```
// main.cpp
```

```
// Week 17 Prog
```

```
//
```

```
// Created by Pippo Pesic on 12/9/22.
```

```
//
```

//This program is an example of the factory method because we create classes derived from Pizza, and then create the objects in a function outside of the class. The main function will not create objects, but rather passes the type of pizza to the create function in order to create the function.

```
#include <stdexcept>
```

```
#include <iostream>
```

```
#include <memory>
```

Philip Pesic

Week 17

December 11 2022

Week 17 Program

```
#include <string>
```

```
using namespace std;
```

```
class Pizza {
```

```
public:
```

```
    virtual double getPrice() const = 0;
```

```
    virtual ~Pizza() { cout << "Destructor called" << endl; }
```

```
};
```

```
class PepperoniOlivePizza : public Pizza {
```

```
public:
```

```
    virtual double getPrice() const { return 8.50; };
```

```
    virtual ~PepperoniOlivePizza() {};
```

```
};
```

```
class DeluxeChickenPizza : public Pizza {
```

```
public:
```

```
    virtual double getPrice() const { return 10.50; };
```

```
    virtual ~DeluxeChickenPizza() {};
```

```
};
```

Philip Pesic

Week 17

December 11 2022

Week 17 Program

```
class HawaiianPizza : public Pizza {  
  
public:  
  
    virtual double getPrice() const { return 11.50; };  
  
    virtual ~HawaiianPizza() {};  
  
};
```

```
class CheesePizza : public Pizza {  
  
public:  
  
    virtual double getPrice() const { return 7.50; };  
  
    virtual ~CheesePizza() {};  
  
};
```

```
class PizzaFactory {  
  
public:  
  
    enum PizzaType {  
  
        PepperoniOlive,  
  
        DeluxeChicken,  
  
        Hawaiian,  
  
        Cheese  
  
    };  
  
};
```

Philip Pesic

Week 17

December 11 2022

Week 17 Program

//Creates a new pizza

```
static unique_ptr<Pizza> createPizza(PizzaType pizzaType) {
    switch (pizzaType) {
        case PepperoniOlive: return make_unique<PepperoniOlivePizza>();
        case DeluxeChicken: return make_unique<DeluxeChickenPizza>();
        case Hawaiian: return make_unique<HawaiianPizza>();
        case Cheese: return make_unique<CheesePizza>();
    }

    throw "invalid pizza type.";
}

};

// Instantiate all available pizzas and print their prices, and names
void pizza_information(PizzaFactory::PizzaType pizzatype)
{
    string PizzaNames[4] = {"PepperoniOlive", "DeluxeChicken", "Hawaiian", "Cheese"};

    unique_ptr<Pizza> pizza = PizzaFactory::createPizza(pizzatype);

    cout << "Price of " << PizzaNames[pizzatype] << " is " << pizza->getPrice() <<
    std::endl;
```

Philip Pesic

Week 17

December 11 2022

Week 17 Program

```
}
```

```
//Call pizza_information for each pizza type
```

```
int main() {
```

```
    pizza_information(PizzaFactory::PepperoniOlive);
```

```
    pizza_information(PizzaFactory::DeluxeChicken);
```

```
    pizza_information(PizzaFactory::Hawaiian);
```

```
    pizza_information(PizzaFactory::Cheese);
```

```
    cout << "Philip Pesic 12/11/22" << endl;
```

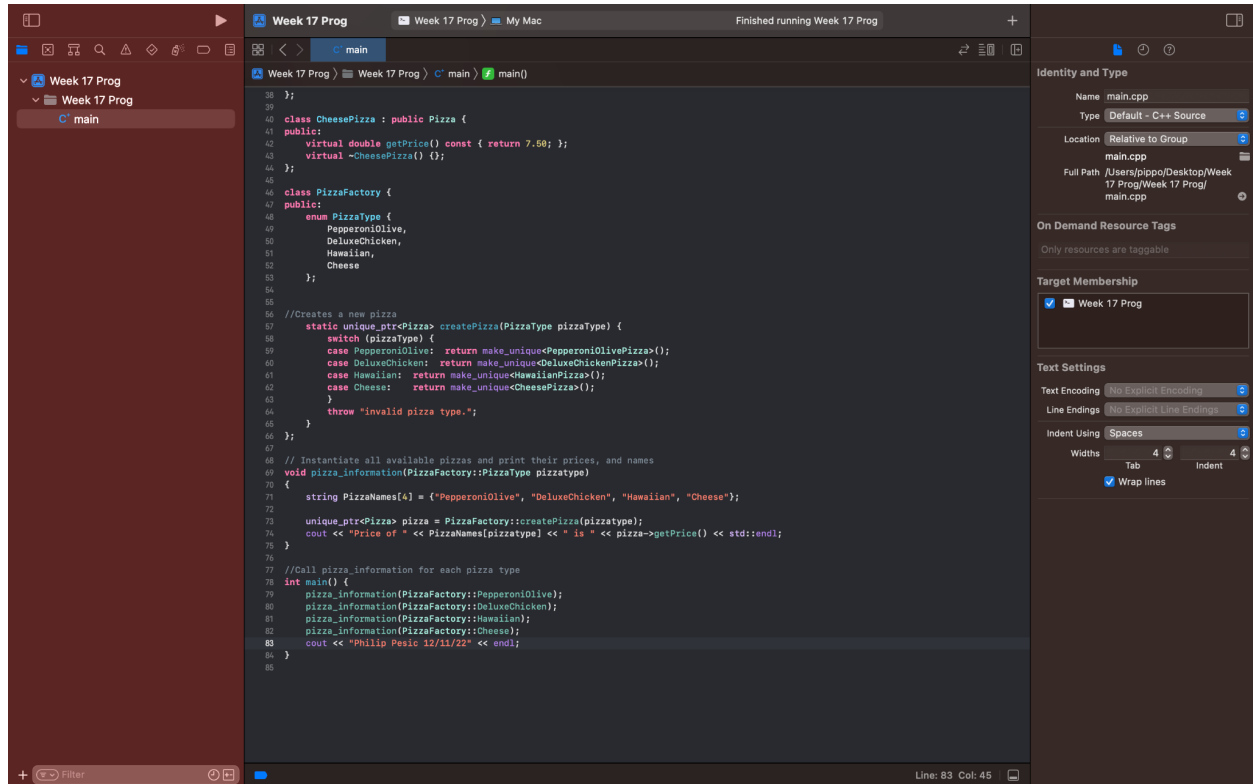
```
}
```

Philip Pesic

Week 17

December 11 2022

Week 17 Program



```
38 };
39
40 class CheesePizza : public Pizza {
41 public:
42     virtual double getPrice() const { return 7.58; };
43     virtual ~CheesePizza() {}
44 };
45
46 class PizzaFactory {
47 public:
48     enum PizzaType {
49         PepperoniOlive,
50         DeluxeChicken,
51         Hawaiian,
52         Cheese
53     };
54 };
55
56 //Creates a new pizza
57 static unique_ptr<Pizza> createPizza(PizzaType pizzaType) {
58     switch (pizzaType) {
59         case PepperoniOlive: return make_unique<PepperoniOlivePizza>();
60         case DeluxeChicken: return make_unique<DeluxeChickenPizza>();
61         case Hawaiian: return make_unique<HawaiianPizza>();
62         case Cheese: return make_unique<CheesePizza>();
63     }
64     throw "invalid pizza type.";
65 }
66 };
67
68 // Instantiate all available pizzas and print their prices, and names
69 void pizza_information(PizzaFactory::PizzaType pizzatype)
70 {
71     string PizzaNames[4] = {"PepperoniOlive", "DeluxeChicken", "Hawaiian", "Cheese"};
72
73     unique_ptr<Pizza> pizza = PizzaFactory::createPizza(pizzatype);
74     cout << "Price of " << PizzaNames[pizzatype] << " is " << pizza->getPrice() << endl;
75 }
76
77 //Call pizza information for each pizza type
78 int main() {
79     pizza_information(PizzaFactory::PepperoniOlive);
80     pizza_information(PizzaFactory::DeluxeChicken);
81     pizza_information(PizzaFactory::Hawaiian);
82     pizza_information(PizzaFactory::Cheese);
83     cout << "Philip Pesic 12/11/22" << endl;
84 }
85
```

Identity and Type

- Name: main.cpp
- Type: Default - C++ Source
- Location: Relative to Group
- Full Path: /Users/jpoo/Desktop/Week 17 Prog/Week 17 Prog/main.cpp

On Demand Resource Tags

- Only resources are taggable

Target Membership

- ☒ Week 17 Prog

Text Settings

- Text Encoding: No Explicit Encoding
- Line Endings: No Explicit Line Endings
- Indent Using: Spaces
- Widths: Tab 4, Indent 4
- ☒ Wrap lines

Line: 83 Col: 45

Philip Pesic

Week 17

December 11 2022

Week 17 Program

The screenshot shows a C++ IDE with a project named "Week 17 Prog". The main.cpp file contains the following code:

```
1 //  
2 // main.cpp  
3 // Week 17 Prog  
4 //  
5 // Created by Pippo Pesic on 12/9/22.  
6 //  
7 //  
8 //This program is an example of the factory method because we create classes derived from Pizza, and then create the objects in a function  
9 //outside of the class. The main function will not create objects, but rather passes the type of pizza to the create function in order  
10 //to create the function.  
11  
12 #include <stdexcept>  
13 #include <iostream>  
14 #include <memory>  
15 #include <string>  
16 using namespace std;  
17  
18 class Pizza {  
19 public:  
20     virtual double getPrice() const = 0;  
21     virtual ~Pizza() { cout << "Destructor called" << endl; }  
22 };  
23  
24 class PepperoniOlivePizza : public Pizza {  
25 public:  
26     virtual double getPrice() const { return 8.50; };  
27     virtual ~PepperoniOlivePizza() {};  
28 };  
29  
30 class DeluxeChickenPizza : public Pizza {  
31 public:  
32     virtual double getPrice() const { return 10.50; };  
33     virtual ~DeluxeChickenPizza() {};  
34 };  
35  
36 class HawaiianPizza : public Pizza {  
37 public:  
38     virtual double getPrice() const { return 11.50; };  
39     virtual ~HawaiianPizza() {};  
40 };  
41  
42 class CheesePizza : public Pizza {  
43 public:  
44     virtual double getPrice() const { return 7.50; };  
45 };  
46  
47 // Factory Method  
48 Pizza* createPizza(string type) {  
49     if (type == "PepperoniOlive") return new PepperoniOlivePizza();  
50     if (type == "DeluxeChicken") return new DeluxeChickenPizza();  
51     if (type == "Hawaiian") return new HawaiianPizza();  
52     if (type == "Cheese") return new CheesePizza();  
53     return nullptr;  
54 }  
55  
56 int main() {  
57     Pizza* p = createPizza("PepperoniOlive");  
58     cout << "Price of PepperoniOlive is " << p->getPrice() << endl;  
59     delete p;  
60     p = createPizza("DeluxeChicken");  
61     cout << "Price of DeluxeChicken is " << p->getPrice() << endl;  
62     delete p;  
63     p = createPizza("Hawaiian");  
64     cout << "Price of Hawaiian is " << p->getPrice() << endl;  
65     delete p;  
66     p = createPizza("Cheese");  
67     cout << "Price of Cheese is " << p->getPrice() << endl;  
68     delete p;  
69     return 0;  
70 }
```

The output window shows the following results:

```
Price of PepperoniOlive is 8.5  
Destructor called  
Price of DeluxeChicken is 10.5  
Destructor called  
Price of Hawaiian is 11.5  
Destructor called  
Price of Cheese is 7.5  
Destructor called  
Philip Pesic 12/11/22  
Program ended with exit code: 0
```

I learned: how to use and implement the factory pattern