User Story:

As a Scrum team member, I want to create a framework that allows for individual applications to define their own domain objects and provide for their instantiation.

Motivation:

Factory Method defines a method, which should be used for creating objects instead of direct constructor call. Subclasses can override this method to change the class of objects that will be created. It's like a Template Method is to implement an algorithm. Factory Method makes a design more customizable and only a little more complicated. Factory Method is a standard way to create objects, but the instantiated object never changes.

Example:

The Factory Method defines an interface for creating objects, but lets subclasses decide which classes to instantiate. Create different types of vehicles that demonstrate this pattern. Manufacturers of vehicles process bike and car. The class of vehicle is determined by the vehicle types.

**Before:**

class pizzaShop

{

public:

virtual void printBill() = 0;

};

class BeefPizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 15 AUD$ \n";

}

};

class VegetablePizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 13 AUD$ \n";

}

};

class CickenPizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 12 AUD$ \n";

}

};

int main()

{

vector<pizzaShop\*> roles;

int choice;

while (true)

{

cout << "BeefPizza(1) VegetablePizza(2) CickenPizza(3) Go(0): ";

cin >> choice;

if (choice == 0)

break;

else if (choice == 1)

roles.push\_back(new BeefPizza);

else if (choice == 2)

roles.push\_back(new VegetablePizza);

else

roles.push\_back(new CickenPizza);

}

for (int i = 0; i < roles.size(); i++)

roles[i]->printBill();

for (int i = 0; i < roles.size(); i++)

delete roles[i];

}

**Mechanics:**

The "factory method" can be defined in the Stooge base class as a factory class. This is a static method of a class that returns an object of that class type. But unlike a constructor, the actual object it returns might be an instance of a subclass. Another advantage of a factory method is that it can return existing instances multiple times.

**After:**

class pizzaShop

{

public:

// Factory Method

static pizzaShop \*makePizza(int choice);

virtual void printBill() = 0;

};

int main()

{

vector<pizzaShop\*> roles;

int choice;

while (true)

{

cout << "BeefPizza(1) VegetablePizza(2) CickenPizza(3) Go(0): ";

cin >> choice;

if (choice == 0)

break;

roles.push\_back(pizzaShop::makePizza(choice));

}

for (int i = 0; i < roles.size(); i++)

roles[i]->printBill();

for (int i = 0; i < roles.size(); i++)

delete roles[i];

}

class BeefPizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 15 AUD$ \n";

}

};

class VegetablePizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 13 AUD$ \n";

}

};

class CickenPizza: public pizzaShop

{

public:

void printBill()

{

cout << "Pirce is 12 AUD$ \n";

}

};

Stooge \*Stooge::make\_stooge(int choice)

{

if (choice == 1)

return new BeefPizza;

else if (choice == 2)

return new VegetablePizza;

else

return new CickenPizza;

}