**Паттерн Abstract Factory (абстрактная фабрика)**

**Назначение паттерна Abstract Factory**

Целесообразно использовать паттерн Abstract Factory если:

• Система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. Непосредственное использование выражения new в коде приложения нежелательно.

• Необходимо создавать группы или семейства взаимосвязанных объектов, исключая возможность одновременного использования объектов из разных семейств в одном контексте.

Примеры групп взаимосвязанных объектов.

Пусть некоторое приложение с поддержкой графического интерфейса пользователя рассчитано на использование на различных платформах, при этом внешний вид этого интерфейса должен соответствовать принятому стилю для той или иной платформы. Например, если это приложение установлено на Windows-платформу, то его кнопки, меню, полосы прокрутки должны отображаться в стиле, принятом для Windows. Группой взаимосвязанных объектов в этом случае будут элементы графического интерфейса пользователя для конкретной платформы.

Другой пример. Рассмотрим текстовый редактор с многоязычной поддержкой, у которого имеются функциональные модули, отвечающие за расстановку переносов слов и проверку орфографии. Если, скажем, открыт документ на русском языке, то должны быть подключены соответствующие модули, учитывающие специфику русского языка. Ситуация, когда для такого документа одновременно используются модуль расстановки переносов для русского языка и модуль проверки орфографии для немецкого языка, исключается. Здесь группой взаимосвязанных объектов будут соответствующие модули, учитывающие специфику некоторого языка.

И последний пример. Рассмотрим игру-стратегию, в которой описывается военное противостояние между армиями Рима и Карфагена. Очевидно, что внешний вид, боевые порядки и характеристики для разных родов войск (пехота, лучники, конница) в каждой армии будут своими. В данном случае семейством взаимосвязанных объектов будут все виды воинов для той или иной противоборствующей стороны, при этом должна исключаться, например, такая ситуация, когда римская конница воюет на стороне Карфагена.

**Описание паттерна Abstract Factory**

Паттерн Abstract Factory реализуется на основе фабричных методов.

Любое семейство или группа взаимосвязанных объектов характеризуется несколькими общими типами создаваемых продуктов, при этом сами продукты таких типов будут различными для разных семейств. Например, для случая стратегической игры общими типами создаваемых продуктов будут пехота, лучники и конница, при этом каждый из этих родов войск римской армии может существенно отличаться по внешнему виду и боевым характеристикам от соответствуюших родов войск армии Карфагена.

Для того чтобы система оставалась независимой от специфики того или иного семейства продуктов необходимо использовать общие интерфейсы для всех основных типов продуктов. В случае стратегической игры это означает, что необходимо использовать три абстрактных базовых класса для каждого типа воинов: пехоты, лучников и конницы. Производные от них классы будут реализовывать специфику соответствующего типа воинов той или иной армии.

Для решения задачи по созданию семейств взаимосвязанных объектов паттерн Abstract Factory вводит понятие абстрактной фабрики. Абстрактная фабрика представляет собой некоторый полиморфный базовый класс, назначением которого является объявление интерфейсов фабричных методов, служащих для создания продуктов всех основных типов (один фабричный метод на каждый тип продукта). Производные от него классы, реализующие эти интерфейсы, предназначены для создания продуктов всех типов внутри семейства или группы. В случае нашей игры базовый класс абстрактной фабрики должен определять интерфейс фабричных методов для создания пехотинцев, лучников и конницы, а два производных от него класса будут реализовывать этот интерфейс, создавая воинов всех родов войск для той или иной армии.

UML-диаграмма классов паттерна Abstract Factory

Реализация паттерна Abstract Factory

Приведем реализацию паттерна Abstract Factory для военной стратегии "Пунические войны". При этом предполагается, что число и типы создаваемых в начале игры боевых единиц идентичны для обеих армий.

**#include <iostream>**

**#include <vector>**

**// Абстрактные базовые классы всех возможных видов воинов**

**class Infantryman**

**{**

**public:**

**virtual void info() = 0;**

**virtual ~Infantryman() {}**

**};**

**class Archer**

**{**

**public:**

**virtual void info() = 0;**

**virtual ~Archer() {}**

**};**

**class Horseman**

**{**

**public:**

**virtual void info() = 0;**

**virtual ~Horseman() {}**

**};**

**// Классы всех видов воинов Римской армии**

**class RomanInfantryman: public Infantryman**

**{**

**public:**

**void info() {**

**cout << "RomanInfantryman" << endl;**

**}**

**};**

**class RomanArcher: public Archer**

**{**

**public:**

**void info() {**

**cout << "RomanArcher" << endl;**

**}**

**};**

**class RomanHorseman: public Horseman**

**{**

**public:**

**void info() {**

**cout << "RomanHorseman" << endl;**

**}**

**};**

**// Классы всех видов воинов армии Карфагена**

**class CarthaginianInfantryman: public Infantryman**

**{**

**public:**

**void info() {**

**cout << "CarthaginianInfantryman" << endl;**

**}**

**};**

**class CarthaginianArcher: public Archer**

**{**

**public:**

**void info() {**

**cout << "CarthaginianArcher" << endl;**

**}**

**};**

**class CarthaginianHorseman: public Horseman**

**{**

**public:**

**void info() {**

**cout << "CarthaginianHorseman" << endl;**

**}**

**};**

**// Абстрактная фабрика для производства воинов**

**class ArmyFactory**

**{**

**public:**

**virtual Infantryman\* createInfantryman() = 0;**

**virtual Archer\* createArcher() = 0;**

**virtual Horseman\* createHorseman() = 0;**

**virtual ~ArmyFactory() {}**

**};**

**// Фабрика для создания воинов Римской армии**

**class RomanArmyFactory: public ArmyFactory**

**{**

**public:**

**Infantryman\* createInfantryman() {**

**return new RomanInfantryman;**

**}**

**Archer\* createArcher() {**

**return new RomanArcher;**

**}**

**Horseman\* createHorseman() {**

**return new RomanHorseman;**

**}**

**};**

**// Фабрика для создания воинов армии Карфагена**

**class CarthaginianArmyFactory: public ArmyFactory**

**{**

**public:**

**Infantryman\* createInfantryman() {**

**return new CarthaginianInfantryman;**

**}**

**Archer\* createArcher() {**

**return new CarthaginianArcher;**

**}**

**Horseman\* createHorseman() {**

**return new CarthaginianHorseman;**

**}**

**};**

**// Класс, содержащий всех воинов той или иной армии**

**class Army**

**{**

**public:**

**~Army() {**

**int i;**

**for(i=0; i<vi.size(); ++i) delete vi[i];**

**for(i=0; i<va.size(); ++i) delete va[i];**

**for(i=0; i<vh.size(); ++i) delete vh[i];**

**}**

**void info() {**

**int i;**

**for(i=0; i<vi.size(); ++i) vi[i]->info();**

**for(i=0; i<va.size(); ++i) va[i]->info();**

**for(i=0; i<vh.size(); ++i) vh[i]->info();**

**}**

**vector<Infantryman\*> vi;**

**vector<Archer\*> va;**

**vector<Horseman\*> vh;**

**};**

**// Здесь создается армия той или иной стороны**

**class Game**

**{**

**public:**

**Army\* createArmy( ArmyFactory& factory ) {**

**Army\* p = new Army;**

**p->vi.push\_back( factory.createInfantryman());**

**p->va.push\_back( factory.createArcher());**

**p->vh.push\_back( factory.createHorseman());**

**return p;**

**}**

**};**

**int main()**

**{**

**Game game;**

**RomanArmyFactory ra\_factory;**

**CarthaginianArmyFactory ca\_factory;**

**Army \* ra = game.createArmy( ra\_factory);**

**Army \* ca = game.createArmy( ca\_factory);**

**cout << "Roman army:" << endl;**

**ra->info();**

**cout << "\nCarthaginian army:" << endl;**

**ca->info();**

**// ...**

**}**

Вывод программы будет следующим:

**Roman army:**

**RomanInfantryman**

**RomanArcher**

**RomanHorseman**

**Carthaginian army:**

**CarthaginianInfantryman**

**CarthaginianArcher**

**CarthaginianHorseman**

**Результаты применения паттерна Abstract Factory**

Достоинства паттерна Abstract Factory

• Скрывает сам процесс порождения объектов, а также делает систему независимой от типов создаваемых объектов, специфичных для различных семейств или групп (пользователи оперируют этими объектами через соответствующие абстрактные интерфейсы).

• Позволяет быстро настраивать систему на нужное семейство создаваемых объектов. В случае многоплатформенного графического приложения для перехода на новую платформу, то есть для замены графических элементов (кнопок, меню, полос прокрутки) одного стиля другим достаточно создать нужный подкласс абстрактной фабрики. При этом условие невозможности одновременного использования элементов разных стилей для некоторой платформы будет выполнено автоматически.

Недостатки паттерна Abstract Factory

• Трудно добавлять новые типы создаваемых продуктов или заменять существующие, так как интерфейс базового класса абстрактной фабрики фиксирован. Например, если для стратегической игры нужно будет ввести новый вид военной единицы - осадные орудия, то надо будет добавить новый фабричный метод, объявив его интерфейс в полиморфном базовом классе AbstractFactory и реализовав во всех подклассах. Снять это ограничение можно следующим образом. Все создаваемые объекты должны наследовать от общего абстрактного базового класса, а в единственный фабричный метод в качестве параметра необходимо передавать идентификатор типа объекта, который нужно создать. Однако в этом случае необходимо учитывать следующий момент. Фабричный метод создает объект запрошенного подкласса, но при этом возвращает его с интерфейсом общего абстрактного класса в виде ссылки или указателя, поэтому для такого объекта будет затруднительно выполнить какую-либо операцию, специфичную для подкласса.