1. Фабричный метод (Factory Method)

- **Цель:** Делегирование создания объектов подклассам.
- **Основная идея:**
  - Вместо создания объектов напрямую в коде, используется фабричный метод, который инкапсулирует логику создания.
  - Позволяет подклассам изменять тип создаваемого объекта.
- **Структура:**
  - Есть **абстрактный класс** с фабричным методом. Подклассы реализуют этот метод и создают конкретные объекты.
- **Когда использовать:**
  - Когда нужно делегировать создание объектов потомкам.
  - Если есть иерархия классов и каждый подкласс должен создавать свои типы объектов.

Пример:

```
class Product {
public:
    virtual void use() = 0;
    virtual ~Product() {}
};

class ConcreteProductA : public Product {
public:
    void use() override { std::cout << "Using Product A\n"; }
};

class ConcreteProductB : public Product {
public:
    void use() override { std::cout << "Using Product B\n"; }
};

class Creator {
public:
    virtual Product* factoryMethod() = 0;
    virtual ~Creator() {}
};

class ConcreteCreatorA : public Creator {
public:
    Product* factoryMethod() override { return new ConcreteProductA(); }
};

class ConcreteCreatorB : public Creator {
public:
    Product* factoryMethod() override { return new ConcreteProductB(); }
};
```

-

---

2. Абстрактная фабрика (Abstract Factory)

- **Цель:** Создание семейств взаимосвязанных объектов.
- **Основная идея:**
    - Позволяет создать группу связанных объектов, не указывая их конкретные классы.
    - Часто используется для обеспечения совместимости создаваемых объектов.
- **Структура:**
    - Есть интерфейс абстрактной фабрики, который определяет методы создания для различных типов объектов.
    - Конкретные фабрики реализуют этот интерфейс и создают конкретные реализации объектов.
- **Когда использовать:**
    - Когда нужно создавать **наборы** объектов, которые должны работать вместе.
    - Если система должна быть независимой от способа создания и представления продуктов.

Пример:

```
class Chair {
public:
    virtual void sitOn() = 0;
};

class ModernChair : public Chair {
public:
    void sitOn() override { std::cout << "Sitting on a modern chair\n"; }
};

class VictorianChair : public Chair {
public:
    void sitOn() override { std::cout << "Sitting on a Victorian chair\n"; }
};

class FurnitureFactory {
public:
    virtual Chair* createChair() = 0;
};

class ModernFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override { return new ModernChair(); }
};

class VictorianFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override { return new VictorianChair(); }
};
```

    -

Основные различия:

Характеристика	Фабричный метод	Абстрактная фабрика
Цель	Создание объектов через подклассы	Создание семейств объектов
Количество продуктов	Один вид продукта	Несколько взаимосвязанных продуктов
Гибкость	Позволяет подклассам определять объекты	Обеспечивает совместимость объектов
Пример	Создание отдельных классов	Создание набора связанных объектов

```
#include <iostream>

// ---------- Фабричный метод ----------

class Product {
public:
   virtual void use() = 0;
   virtual ~Product() {}
};

class ConcreteProductA : public Product {
public:
   void use() override { std::cout << "Using Product A\n"; }
};

class ConcreteProductB : public Product {
public:
   void use() override { std::cout << "Using Product B\n"; }
};

class Creator {
public:
   virtual Product* factoryMethod() = 0;
   virtual ~Creator() {}
};

class ConcreteCreatorA : public Creator {
public:
   Product* factoryMethod() override { return new ConcreteProductA(); }
};

class ConcreteCreatorB : public Creator {
public:
   Product* factoryMethod() override { return new ConcreteProductB(); }
};

// ---------- Абстрактная фабрика ----------

// Интерфейс стула
class Chair {
public:
   virtual void sitOn() = 0;
```

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

class ModernChair : public Chair {
public:
    void sitOn() override { std::cout << "Sitting on a modern chair\n"; }
};

class VictorianChair : public Chair {
public:
    void sitOn() override { std::cout << "Sitting on a Victorian chair\n"; }
};

// Интерфейс стола
class Table {
public:
    virtual void eatOn() = 0;
    virtual ~Table() {}
};

class ModernTable : public Table {
public:
    void eatOn() override { std::cout << "Eating on a modern table\n"; }
};

class VictorianTable : public Table {
public:
    void eatOn() override { std::cout << "Eating on a Victorian table\n"; }
};

// Абстрактная фабрика мебели
class FurnitureFactory {
public:
    virtual Chair* createChair() = 0;
    virtual Table* createTable() = 0;
    virtual ~FurnitureFactory() {}
};

// Конкретная фабрика для модерн-стиля
class ModernFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override { return new ModernChair(); }
    Table* createTable() override { return new ModernTable(); }
};

// Конкретная фабрика для викторианского стиля
class VictorianFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override { return new VictorianChair(); }
    Table* createTable() override { return new VictorianTable(); }
};

// ---------- Тестирование ----------

int main() {
    // Используем Фабричный метод
    Creator* creatorA = new ConcreteCreatorA();
    Product* productA = creatorA->factoryMethod();
    productA->use();
    delete productA;
    delete creatorA;
```

```
    // Используем Абстрактную фабрику
    FurnitureFactory* factory = new ModernFurnitureFactory();
    Chair* chair = factory->createChair();
    Table* table = factory->createTable();

    chair->sitOn();
    table->eatOn();

    delete chair;
    delete table;
    delete factory;

    return 0;
}
```