**Лабораторная работа №1**

**«Реализация одного из порождающих паттернов проектирования»**

**Цель работы:** научиться применять порождающие паттерны проектирования.

**Продолжительность работы** - 4 часа.

**Содержание**

[Порождающие паттерны 1](#_Toc99797138)

[Паттерн Абстрактная фабрика 1](#_Toc99797139)

# **Порождающие паттерны**

Порождающие паттерны проектирования абстрагируют процесс инстанцирования объектов. Они позволяют сделать код независимым от способа создания, композиции и представления используемых в его работе объектов.

Список порождающих паттернов (GoF):

* Фабричный метод (*Factory method*)
* Абстрактная фабрика (*Abstract Factory*)
* Строитель (*Builder*)
* Прототип (*Prototype*)
* Одиночка (*Singleton*)

# **Паттерн Абстрактная фабрика**

**Назначение**

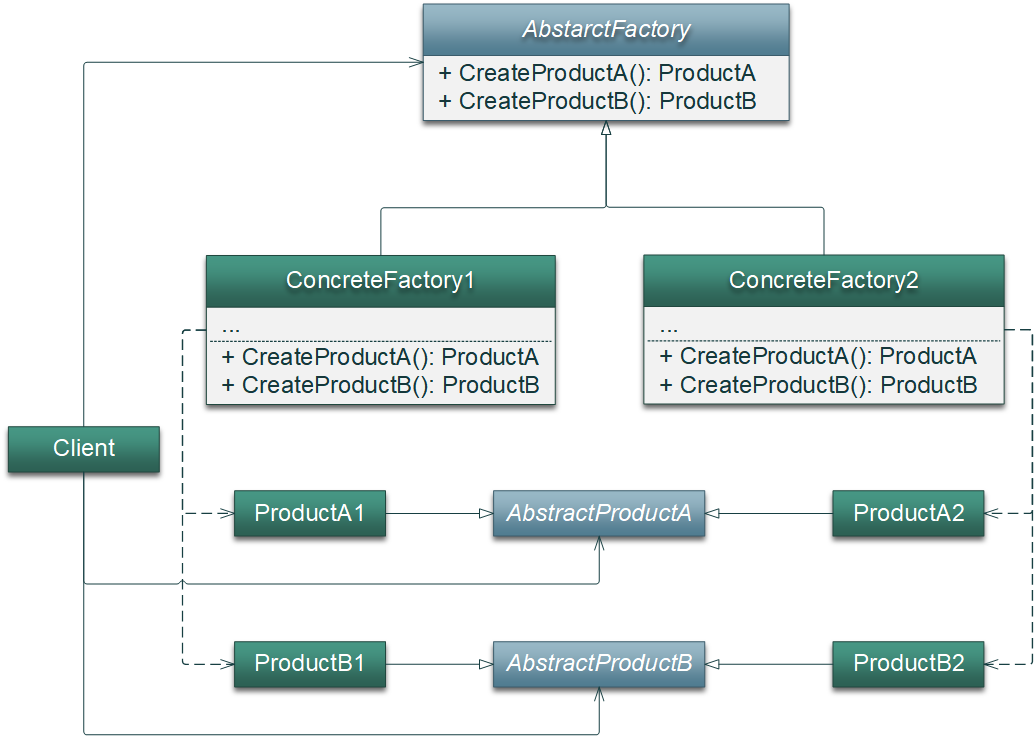
Абстрактная фабрика – паттерн, порождающий объекты. Этот паттерн позволяет работать с разными видами связанных друг с другом продуктов, не завися от конкретных продуктов. Он скрывает от клиентского кода подробности того, какие конкретные объекты будут созданы. Клиент будет работать со всеми типами конкретными продуктов через заранее определённый для них общих интерфейс.

**Применимость**

Использование паттерна Abstract Factory (абстрактная фабрика) целесообразно если:

* Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты
* Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения.
* Система должна конфигурироваться одним из семейств составляющих ее объектов, а вы хотите предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.
* Когда класс содержит много фабричных методов, чтобы он не потерял свою основную цель, рекомендуется перенести логику создания продуктов в отдельную иерархию классов и применить Абстрактную фабрику. *Один класс должен решать только какую-то одну задачу, такой принцип программирования называется принципом единственной ответственности (Single Responsibility Principle, SRP)*

**Структура**



Участники

* ***AbstractFactory*** - абстрактная фабрика: объявляет интерфейс для операций, создающих абстрактные объекты-продукты
* **ConcreteFactory** (ConcreteFactory1, ConcreteFactory2) - конкретная фабрика: реализует операции, создающие конкретные объекты-продукты
* ***AbstractProduct*** (*AbstractProductА*, *AbstractProductВ*) - абстрактный продукт: объявляет интерфейс для типа объекта-продукта
* **ConcreteProduct** (ProductА, ProductВ) - конкретный продукт: определяет объектпродукт, создаваемый соответствующей конкретной - реализует интерфейс *AbstractProduct*
* **Client** - клиент: пользуется исключительно интерфейсами, которые объявлены в классах *AbstractFactory* и *AbstractProduct*

**Примеры использования**

* Нужно выпускать машины с разным типом кузова, конкретными фабриками будут производители Toyota, Kia и Lada, а их конкретные продукты – седан, купе и универсал.
* Нужно выпускать одежду разного типа, конкретными фабриками будут производители ZARA, H&M и Bershka, а их конкретные продукты – верхняя одежда, нижняя и головные уборы.
* Когда делаете кроссплатформенное приложение, то для правильного отображения UI элементов на разных устройствах нужно использовать стиль и настройки, которые подходят именно этой платформе.

**Пример кода для *Abstract Factory***

public abstract class AbsractDatabase

{

private string \_connectionString;

public AbsractDatabase(string connectionString)

{

\_connectionString = connectionString;

Console.WriteLine($"Connetion to {\_connectionString}");

}

public abstract List<AbstractTable> GetTables();

public abstract List<AbstractQuery> GetQueries();

}

public abstract class AbstractTable

{

protected string \_tableName;

protected string[] \_columnNames;

protected int \_numberRecords;

public AbstractTable(string tableName, string[] columnNames,

int numberRecords)

{

\_tableName = tableName;

\_columnNames = columnNames;

\_numberRecords = numberRecords;

}

public abstract void PrintInfo();

}

public abstract class AbstractQuery

{

protected string \_name;

protected string \_query;

public AbstractQuery(string name, string query)

{

\_name = name;

\_query = query;

}

public abstract void Execute();

}

public class SqlServerTable : AbstractTable

{

public readonly string Owner;

public SqlServerTable(string tableName, string[] columnNames,

int numberRecords, string owner)

: base(tableName, columnNames, numberRecords)

{

Owner = owner;

}

public override void PrintInfo()

{

Console.WriteLine($"I'am MS SQL Server table," +

$" {\_tableName}," +

$" and my owner is {Owner}" +

$"\nMy columns:");

foreach (string column in \_columnNames)

{

Console.Write(column + " / ");

}

Console.WriteLine();

Console.WriteLine($"Number records - {\_numberRecords}");

}

}

public class SqliteTable : AbstractTable

{

public SqliteTable(string tableName, string[] columnNames,

int numberRecords)

: base(tableName, columnNames, numberRecords) { }

public override void PrintInfo()

{

Console.WriteLine($"I'am SQLite table, {\_tableName}" +

$"\nMy columns:");

foreach(string column in \_columnNames)

{

Console.Write(column + " | ");

}

Console.WriteLine();

Console.WriteLine($"Number records - {\_numberRecords}");

}

}

public class SqlServerQuery : AbstractQuery

{

private int \_executionTime;

public SqlServerQuery(string name, string query)

: base(name, query) { }

public override void Execute()

{

\_executionTime = 300;

Thread.Sleep(300); // Имитация выполнения запроса

Console.WriteLine($"SqlServer Query {\_name} was executed," +

$" Execution time = {\_executionTime}");

}

}

public class SqliteQuery : AbstractQuery

{

public SqliteQuery(string name, string query)

: base(name, query) { }

public override void Execute()

{

Thread.Sleep(500); // Имитация выполнения запроса

Console.WriteLine($"Sqlite Query {\_name} was executed");

}

}

public SqlServerDatabase(string connectionString)

: base(connectionString) { }

public override List<AbstractQuery> GetQueries()

{

List<AbstractQuery> queries = new List<AbstractQuery>();

queries.Add(new SqlServerQuery("Get animals",

"SELECT \* FROM Animals"));

queries.Add(new SqlServerQuery("Get animal with Id = 25",

"SELECT \* FROM Animals" +

"WHERE Id = 1"));

queries.Add(new SqlServerQuery(@"Get bunnies",

"SELECT \* FROM Animals" +

"WHERE type = 'bunny'"));

return queries;

}

public override List<AbstractTable> GetTables()

{

List<AbstractTable> tables = new List<AbstractTable>();

tables.Add(new SqlServerTable("Animals",

new string[] { "Id", "Name", "Type" }, 8, "Ivan")

);

return tables;

}

public SqliteDatabase(string connectionString)

: base(connectionString) { }

public override List<AbstractQuery> GetQueries()

{

List<AbstractQuery> queries = new List<AbstractQuery>();

queries.Add(new SqliteQuery("Get users",

"SELECT \* FROM Users"));

queries.Add(new SqliteQuery("Delete user with Id = 1",

"DELETE FROM Users" +

"WHERE Id = 1"));

return queries;

}

public override List<AbstractTable> GetTables()

{

List<AbstractTable> tables = new List<AbstractTable>();

tables.Add(new SqliteTable("Users",

new string[] { "Id", "First name", "Last name"}, 15)

);

tables.Add(new SqliteTable("Countries",

new string[] { "Id", "Name", "Сapital" }, 60)

);

return tables;

}

static void Main(string[] args)

{

string connectionString = "Server=(localdb)\\" +

"mssqllocaldb;Database=helloappdb";

AbsractDatabase db =

new SqlServerDatabase(connectionString);

List<AbstractTable> tables = db.GetTables();

Console.WriteLine("Tables");

foreach (var table in tables)

{

table.PrintInfo();

}

List<AbstractQuery> queries = db.GetQueries();

Console.WriteLine("Execute queries");

foreach (var query in queries)

{

query.Execute();

}

//--------------------------------------------------

Console.WriteLine();

connectionString = "Data Source=Another.db";

db = new SqliteDatabase(connectionString);

tables = db.GetTables();

Console.WriteLine("Tables");

foreach (var table in tables)

{

table.PrintInfo();

}

queries = db.GetQueries();

Console.WriteLine("Execute queries");

foreach (var query in queries)

{

query.Execute();

}

}

}