**Адаптер**

# **Назначение и цель применения паттерна**

**Адаптер** — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Часто в новом программном проекте не удается повторно использовать уже существующий код. Например, имеющиеся классы могут обладать нужной функциональностью, но иметь при этом несовместимые интерфейсы. В таких случаях следует использовать паттерн Adapter.

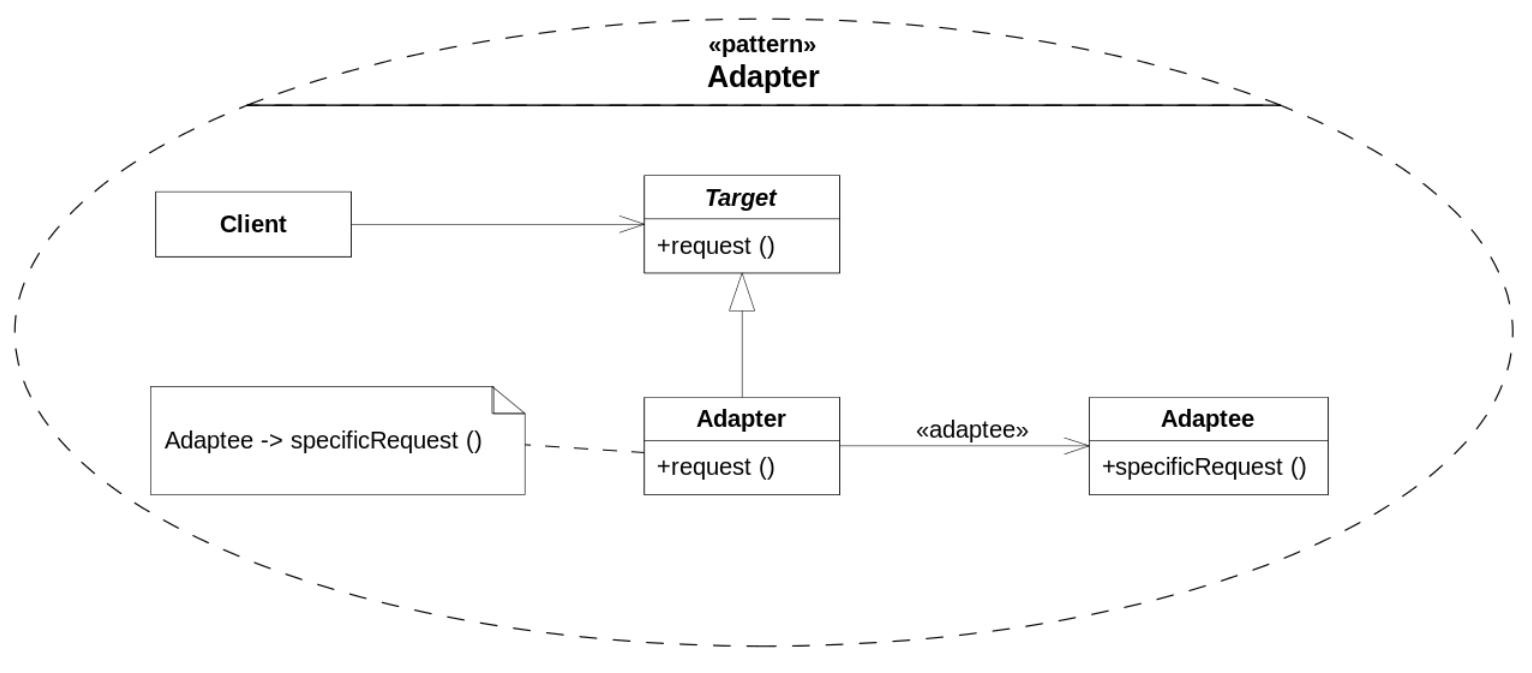
**Проблема**: обеспечить взаимодействие объектов с различными интерфейсами. Адаптировать, а не переписывать существующий код к требуемому интерфейсу.

**Основное назначение**: Паттерн Adapter, представляющий собой программную обертку над существующими классами, преобразует их интерфейсы к виду, пригодному для последующего использования.

**Цель применения паттерна**: преобразование интерфейса одного класса в интерфейс другого. Благодаря реализации данного паттерна мы можем использовать вместе классы с несовместимыми интерфейсами.

# **UML диаграммы паттерна**

Для описания структуры паттерна Interpreter необходимо построить UML диаграмму классов.



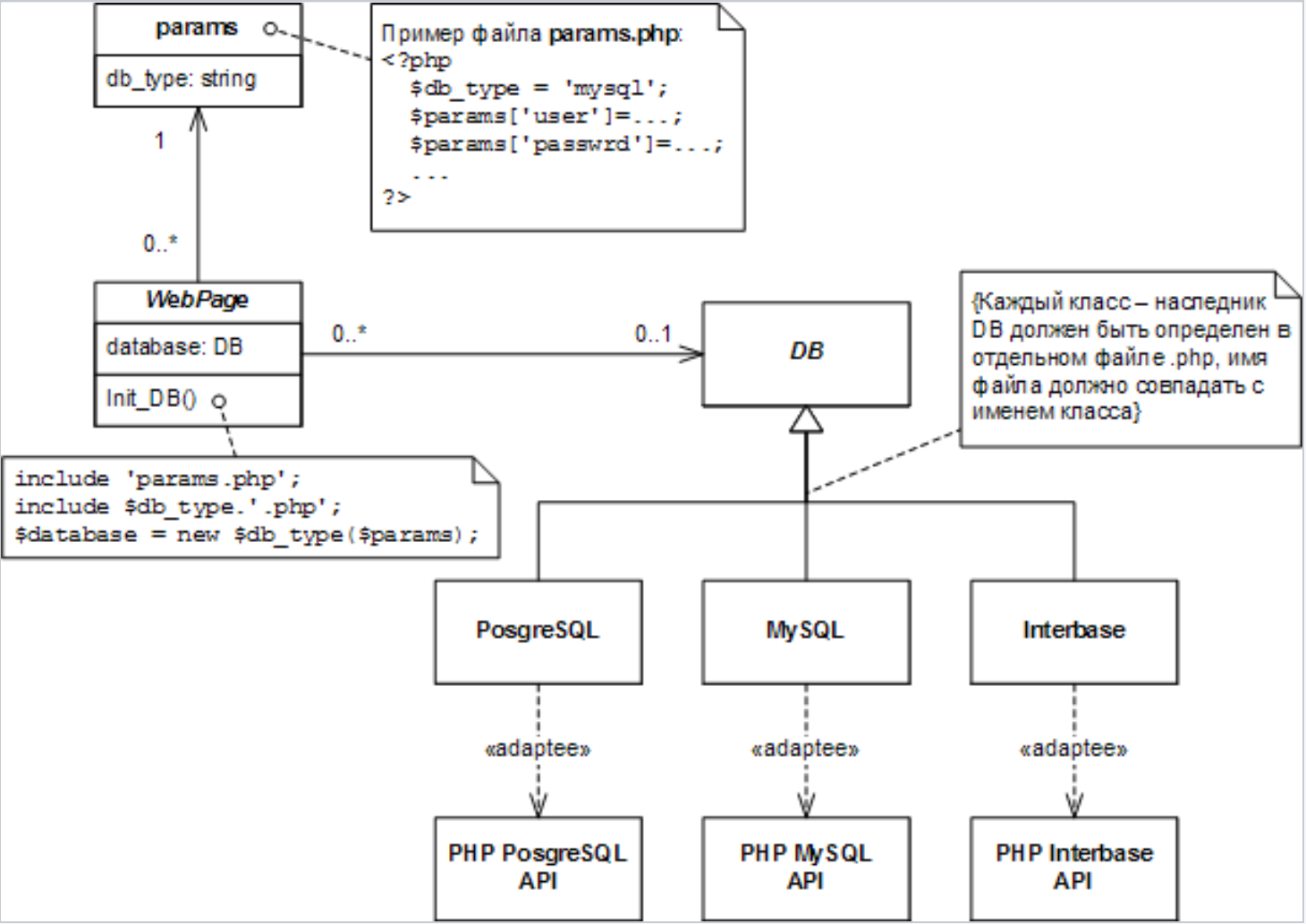
Основные элементы диаграммы:

* + - 1. **Класс Адаптируемый (Adaptee)** является классом, интерфейс которого необходимо адаптировать к целевому виду.
      2. **Класс Целевой (Target)** является классом, к интерфейсу которого необходимо привести адаптируемый класс.
      3. **Класс Адаптер (Adapter)** является классов, который адаптирует интерфейс класса **Адаптируемый** к интерфейсу класса **Целевой**.
      4. **Объект Клиент (Client)** – это объект, который будет использовать объект класса **Адаптируемый** (посредством класса **Адаптер)** так, словно данный класс является экземпляром класса **Целевой**.

# **Области применения**

Типичным примером использования шаблона Адаптер можно назвать создание классов, приводящих к единому интерфейсу функции языка PHP обеспечивающие доступ к различным СУБД.

Вариант решения данной проблемы с использованием шаблона Адаптер показан на рисунке:



# **Особенности паттерна**

Достоинства паттерна Adapter

* Паттерн Adapter позволяет повторно использовать уже имеющийся код, адаптируя его несовместимый интерфейс к виду, пригодному для использования.

Недостатки паттерна Adapter

* Задача преобразования интерфейсов может оказаться непростой в случае, если клиентские вызовы и (или) передаваемые параметры не имеют функционального соответствия в адаптируемом объекте.

# **Пример реализации**

Была разработана программа, которая адаптирует показания термометров, измеряющих температуру по шкалам Фаренгейта и Кельвина в градусы Цельсия. Для этого были выполнены следующие действия.

class Celsi

{

protected double temperature;

public virtual double GetTemperature()

{

if (temperature >= -90 && temperature <= 60)

return temperature;

else

return 0;

}

public virtual void SetTemperature(double t)

{

if (t >= -90 && t <= 60)

temperature = t;

else temperature = 0;

}

public Celsi()

{

temperature = 0;

}

public Celsi(double t)

{

temperature = t;

}

}

Был создан класс Celsi, который позволяет устанавливать и возвращать температуру в градусах Цельсия. Здесь данный класс выступает в качестве **Целевого** **(Target)**.

class Fahrenheit

{

double fah;

public double GetFahrenheit()

{

if (fah >= -130 && fah <= 140)

return fah;

else return 32;

}

public void SetFahrenheit(double f)

{

if (f >= -130 && f <= 140)

fah = f;

else fah = 32;

}

public Fahrenheit()

{

fah = 32;

}

public Fahrenheit(double f)

{

fah = f;

}

}

Класс Fahrenheit позволяет аналогичным образом устанавливать и получать температуру в Фаренгейтах. Здесь класс выступает как класс **Адаптируемый (Adaptee)**.

class Kalvin

{

double kal;

public double GetKalvin()

{

if (kal >= 183.15 && kal <= 333.15)

return kal;

else return 273.15F;

}

public void SetKalvin(double k)

{

if (k >= 183.15F && k <= 333.15F)

kal = k;

else kal = 273.15F;

}

public Kalvin()

{

kal = 273.15F;

}

public Kalvin(double k)

{

kal = k;

}

}

Класс Kalvin позволяет аналогичным образом устанавливать и получать температуру в Кельвинах. Этот класс аналогично выступает как класс **Адаптируемый (Adaptee)**.

class Adapter : Celsi

{

private Kalvin kalvin = new Kalvin();

private Fahrenheit fahrenheit = new Fahrenheit();

public Adapter(Kalvin k)

{

kalvin = k;

fahrenheit = null;

}

public Adapter(Fahrenheit f)

{

fahrenheit = f;

kalvin = null;

}

public override double GetTemperature()

{

if (kalvin != null)

{

return kalvin.GetKalvin() - 273.15F;

}

if (fahrenheit != null)

{

return (fahrenheit.GetFahrenheit() - 32) \* 5 / 9;

}

return 0;

}

public override void SetTemperature(double t)

{

if (kalvin != null)

{

kalvin.SetKalvin(t);

}

else

if (fahrenheit != null)

{

fahrenheit.SetFahrenheit(t);

}

else

temperature = 0;

}

}

Данный класс Adapter выступает в качестве адаптера, который преобразует температуру в Фаренгейтах и Кельвинах в градусы Цельсия.

class Program

{

static Random rnd = new Random();

static void Main(string[] args)

{

Console.WriteLine("Погода в течении апреля:\n");

for(int i = 1; i <= 30; i++)

{

Celsi todayTemp = new Celsi();

switch(i % 3)

{

case 0:

todayTemp = new Celsi();

todayTemp.SetTemperature(rnd.Next(-3, 15) + rnd.NextDouble());

break;

case 1:

todayTemp = new Adapter(new Kalvin());

todayTemp.SetTemperature(rnd.Next(268, 289) + rnd.NextDouble());

break;

case 2:

todayTemp = new Adapter(new Fahrenheit());

todayTemp.SetTemperature(rnd.Next(23, 59) + rnd.NextDouble());

break;

}

Console.WriteLine(i + "го апреля температура: " + Math.Round(todayTemp.GetTemperature(),1)+"\n");

}

Console.Read();

}

}

Здесь с помощью генератора случайных чисел выводится получается температура в градусах Цельсия исходя из различных входных данных. Здесь каждые три дня в течении апреля происходит поочередное измерение температуры 3-мя разными термометрами. При этом выходной результат – всегда в градусах Цельсия. Пример работы программы приведен ниже:

