12. Обобщенные или параметризованные классы

Обобщенный класс – это класс, при описании свойств и методов которого вместо **конкретных типов данных** можно использовать **условные** заменители - обобщенные типы данных.

Количество используемых обобщенных типов при описании класса может быть любым (чаще всего – один). Каждый обобщенный тип имеет условное **имя**, которое и используется при описании полей данных, параметров методов и возвращаемых значений. На основе **одного** обобщенного класса можно создавать объекты с **РАЗНЫМИ** типами данных.

Для замены обобщенных типов конкретными типами данных при **описании объектов** надо указать эти типы. Тип элемента данных можно рассматривать как параметр класса и поэтому такие классы можно называть **параметризованными.** Это поднимает разработку объектных программ на еще более высокий уровень абстракции. Например, вместо **нескольких** реализаций контейнера для **разных** типов данных можно создать **один обобщенный** контейнер, на основе которого генерировать контейнеры для конкретных типов данных.

Преимущества использования обобщенных классов:

- сокращение объема исходного кода за счет использования более общих конструкций
- возможность создания более надежных программ за счет использования механизма строгого контроля типов

Среди всех объектных языков наиболее долго обобщенные классы используются в языке C++. В этом языке обобщенными могут объявляться не только классы, но и их методы и просто обычные функции. Для этого в языке было введено специальное понятие «**шаблон**» (**template**). Использование шаблонов позволяет существенно сократить объем исходного программного кода, переложив часть работы на компилятор.

Многолетняя удачная практика использования шаблонов при написании программ на C++ привела к тому, что аналогичные идеи были воплощены и в других объектных языках. В частности, в язык Java начиная с версии 1.5 включено понятие обобщенных (**generic**) классов. Аналогичный механизм был включен в язык C# и в целом в платформу .NET Framework. Также механизм обобщенных классов реализован в поздних версиях пакета Delphi.

Необходимо отметить, что, несмотря на общность самой идеи обобщенных классов, реализация этого механизма **различна** в разных языках. Особенно сильным это различие является для языка C++ с его template-шаблонами и остальными языками с их generic-классами.

Более подробное рассмотрение обобщенных классов начнем с языков C# и Java. Для объявления обобщенного класса необходимо в его **заголовке** ввести имена используемых в нем условных типов. Эти имена могут быть любыми разрешенными, но исторически принято использовать имена **T1**, **T2**, **T3** и т.д. Синтаксически эти имена заключаются в угловые скобки после имени класса.

Пример описания обобщенного класса с **двумя** условными типами:

```
class   MyGener <T1, T2 >      // заголовок обобщенного класса
{ private   T1   Data1;    // поле данных условного типа  T1
  private   T2   Data2;    // поле данных другого условного типа  T2

  public  MyGener(T1  aD1, T2   aD2) { Data1= aD1;  Data2 = aD2;}
                          // конструктор с двумя обобщенными параметрами

  public  void  SetData1(T1  aD1) { Data1 = aD1 ;} // метод с обобщ. парам.
  public   T2   GetData2( ) {  };  // метод с выходным обобщенным типом
};
```

Из примера видно, что описание обобщенного класса мало чем отличается от описания обычных классов. Самое интересное начинается на

этапе **создания** объектов с помощью обобщенных классов: можно генерировать объекты с **любыми различными** парами типов, как стандартными так и введенными самим пользователем! Для этого достаточно при объявлении объектной переменной после имени обобщенного класса указать (опять же в угловых скобках) имена конкретных типов. А в операторе создания объекта после имени конструктора еще раз повторить эту комбинацию.

Например, для создания объекта на основе класса MyGener с полями **целого** и **вещественного** типа необходимы следующие конструкции:

```
MyGener <int, float>  MyObj1;  // объявление с конкретизацией типов
MyObj1 = new  MyGener <int, float> (100, 3.14);
                                  // создание с конкретизацией типов
```

А вот – объект того же класса, но с **другими** полям стандартного типа:

```
MyGener <string, bool>  MyObj2;
MyObj2 = new  MyGener <string, bool> ('Hello', true);
```

Вместо стандартных типов точно так же можно использовать собственные, лишь бы они были предварительно объявлены:

```
MyGener <MyType1,  MyType2>  MyObj3;
MyObj3 = new  MyGener <MyType1, MyType2> ( );
```

При обработке этих конструкций компилятор проверит соответствие типов и при необходимости выдаст сообщение об ошибке.

Пример **конкретного** и практически полезного обобщенного класса – стек на основе массива (объявлено только самое необходимое с точки зрения текущего материала, остальные очевидные элементы класса отсутствуют).

```
class  GenStack<T>   //  здесь T – условное имя для типа элементов стека
{  private   T[ ]   Arr;   //  объявление массива элементов условного типа  T
   private   int   sp;
   public   void   Push (T   aItem) { …; Arr[sp] = aItem; }
   public   T   Pop( ) { … ; return  Arr[sp]; }
```

```
};
```

Создание и использование объекта-стека для текстовых строк:

```
GenStack<string>  StackString = new  GenStack<string> ( );
StackString.Push ('First string');
StackString.Push ('Second string');
```

А теперь - создание и использование объекта-стека для объектов класса Student (он должен быть объявлен):

```
GenStack<Student>  StackStud = new  GenStack< Student > ( );
StackStud.Push (Stud1);
StackStud.Push (Stud2);
```

Динамическое создание объектов необходимого типа происходит под управлением среды поддержки выполнения программ (Java Virtual Machine или .NET Common Language Runtime).

Весьма похоже выполняется описание и использование обобщенных классов в языке Delphi. Пример - рассмотренный выше класс стеков на основе массива.

Шаг 1. Описание обобщенного класса:

```
TGenStack<T> = class     // T – условное имя для типа элементов стека
  private
    Arr : array of T;  // объявление массива элементов условного типа T
    sp : integer;
  public
    procedure Push (aItem : T);  // метод с обобщенным параметром
    function Pop : T;  // функция обобщенного типа
end;
```

Шаг 2. Реализация методов с использованием условного типа (как обычно, в заголовке метода указывается имя класса, но теперь – с добавлением имени условного типа):

```
procedure TGenStack<T>.Push (aItem : T);
```

begin . . . Arr[sp] := aItem; . . . **end**;

function TGenStack<T>.Pop : T;

begin . . . **result** := Arr[sp]; . . . **end**;

Шаг 3. Объявление объектов-стеков для **конкретных** типов данных:

var StringStack : TGenStack<**string**>; // стек для строк

StudStack : TGenStack<TSudent>; // стек объектов-студентов

Шаг 4. Создание и использование созданных объектов для обработки **конкретных** данных:

StringStack := TGenStack<**string**>.Create; // создание объекта-стека строк

StudStack := TGenStack<TSudent>.Create; // создание стека студентов

StringStack.Push (“First string”);

StringStack.Push (“Second string”);

writeln (StringStack.Pop);

StudStack.Push (Stud1);

StudStack.Push (Stud2);

Stud3 := StudStack.Pop;

В заключение кратко рассмотрим особенности использования **шаблонов** языка C++. Прежде всего отметим более сложную структуру заголовка шаблонного класса, где кроме стандартной директивы **class** используется дополнительная директива **template**:

template <**class** T> **class** MyTemplateClass

Само тело класса описывается уже привычным образом, т.е. с использованием имени условного типа. Еще одна синтаксическая особенность возникает при описании программной реализации методов, вынесенной за пределы класса. Опять же достаточно громоздко приходится описывать заголовки таких методов как обобщенных функций. Например, заголовок конструктора выглядит так:

template <**class** T> MyTemplateClass<T>:: MyTemplateClass(…)

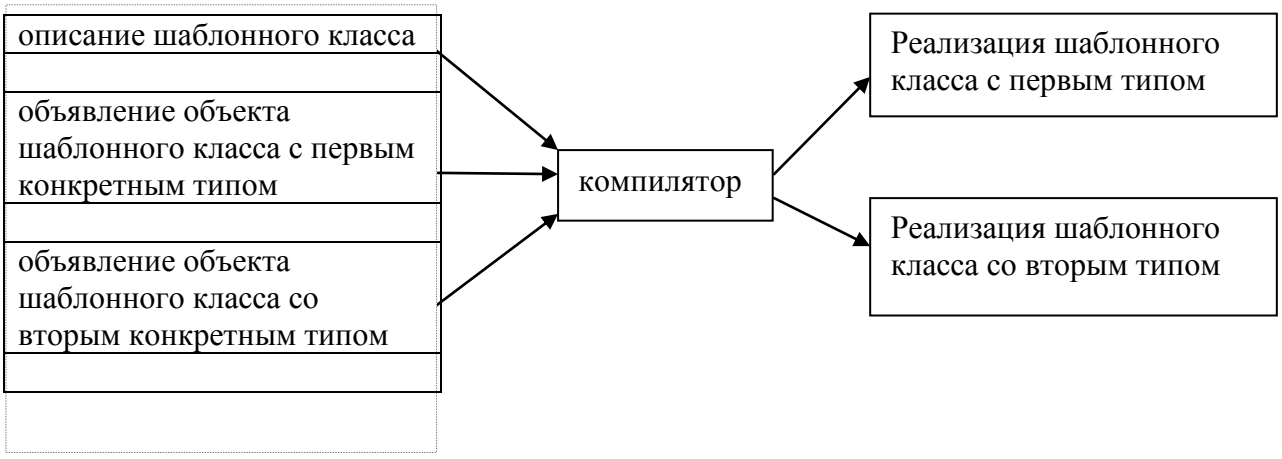
а заголовок метода доступа для чтения поля данных типа T – так:

template <**class** T> T MyTemplateClass<T>:: GetData(…)

После описания шаблонного класса и реализации его методов можно объявлять переменные-экземпляры и именно в этот момент выполняется замена условного типа конкретным:

```
MyTemplateClass <int>   MyIntObj;  // объект с целочисленным полем
MyTemplateClass <float>   MyFloatObj; // объект с вещественным полем
MyTemplateClass <Student>  MyStud;     //  объект со полем типа Student
```

Эта замена выполняется **компилятором,** который для каждой объявленной разновидности создает **свой** отдельный вариант реализации класса. Потом уже на основе этой конкретной реализации создаются необходимые объекты. В этом состоит принципиальное отличие шаблонов языка С++ от обобщенных классов других языков: переход от обобщенных классов к конкретным выполняется **на этапе компиляции** программы, а не на этапе ее выполнения!



Исходный текст программы

Большим достоинством языка С++ является наличие мощной встроенной **библиотеки** стандартных шаблонов **STL** (Standard Template Library). Эта библиотека содержит, в частности, целый ряд шаблонных контейнеров (Vector, List, Stack, Queue, Map). Данные классы позволяют создавать контейнеры для обработки объектов любых необходимых типов.