Лекція 16

# Характеристики (traits) і стратегії (policies)

За визначенням, клас характеристик — це клас, що інкапсулює набір типів і функцій, необхідних шаблонним класам і функціям для маніпулювання об'єктами, тип яких визначається шаблонними параметрами. Типовим прикладом характеристик є класи `char_traits`, що описують властивості символів, і `iterator_traits`, що визначають категорію итератора в бібліотеці STL.

> ***Клас характеристик (властивостей)*** *— це клас, що інкапсулює набір типів і функцій, необхідних шаблонним класам і функціям для маніпулювання об'єктами, тип яких визначається шаблонними параметрами.*

## 17.1. Настроювання алгоритмів за допомогою характеристик

Розглянемо приклад, у якому клас характеристик використовується для уточнення алгоритму. Допустимо, необхідно розробити шаблонну функцію `mean()`, що обчислює середнє арифметичне чи середнє геометричне на основі заданих властивостей.

Напишемо спочатку шаблонні функції, що обчислюють середнє арифметичне і середнє геометричне.

**Обчислення середнього арифметичного**

```
#include <iostream>

 using namespace std;

 template <typename T>
 T mean(T const* array, int N)
 {
    T result = T(0); // Лічильник ініціалізується нулем
    for(int i=0; i<N; i++) {result += *array; array++;}
    return result/N;
 };

int main()
 {
    double array[] = { 1.2, 3.4, 5.6, 7.8, 9.0};
    cout << mean(array,5);
 }
```

**Обчислення середнього геометричного**

```
#include <iostream>

 using namespace std;

 template <typename T>
 T mean(T const* array, int N)
 {
    T result = T(1); // Лічильник ініціалізується одиницею
    for(int i=0; i<N; i++) {result *= *array; array++;}
    return exp(log(result)/N);
 };



int main()
 {
    double array[] = { 1.2, 3.4, 5.6, 7.8, 9.0};
    cout << mean(array,5);
 }
```

Вихідними даними для обчислення как середнього арифметичного, так і середнього геометричного є масив і его розмір. Відмінності полягають у способі ініціалізації і нагромадження даних: при обчисленні середнього арифметичного дані сумуються, а при обчисленні середнього геометричного — перемножуються. Відповідно, у першому випадку лічильник випливає ініціалізувати нулем, а в другому — одиницею. Саме ці дві властивості і є *характеристиками* алгоритму, що залежать від типу даних і поставленої задачі.

Крім того, конкретизація шаблонної функції `mean()` цілочисельними аргументами може привести до виходу за межі припустимого діапазону. Отже, необхідно передбачити автоматичне розширення типу при створенні окремих спеціалізацій, наприклад `char- >long`, `int- >long` і `float- >double`.

Спробуємо врахувати ці тонкості, скориставшись класом характеристик `MeanTraits` і механізмом часткової спеціалізації.

**Настроювання обчислення середнього арифметичного**

```
#include <iostream>

using namespace std;

template<typename T>
class MeanTraits
{
public:
   typedef T MeanType;
   static MeanType start()
      {  return 0; }
};

template<>
class MeanTraits<char>
{
public:
   typedef int MeanType;
   static MeanType start()
      {  return 0; }
};

template<>
class MeanTraits<int>
{
public:
   typedef long MeanType;
   static MeanType start()
     {  return 0; }
};

template<>
class MeanTraits<float>
{
public:
   typedef double MeanType;
   static MeanType start()
     {  return 0.0; }
};

template <typename T>
typename MeanTraits<T>::MeanType mean(T const* array, int N)
{
   typedef typename MeanTraits<T>::MeanType MeanType;
   MeanType result = MeanTraits<T>::start();

   for(int i=0; i<N; i++) {result += *array; array++;}
   return result/N;
};
```

```
int main()
{
   int array[] = { 1, 3, 5, 7, 9};
   cout << mean(&array[0],5);
}
```

Виділимо несколько принципово важливих моментів. По-перше, просування типів здійснюється за допомогою часткової спеціалізації шаблонів. Кожна часткова спеціалізація (клас характеристик) містить перевизначення типу результату і визначення функції, що обчислює початкове значення лічильника. По-друге, початкове значення лічильника задається *статичною функцією*. Для цієї мети можна було б скористатися статичною константою, але з нею зв'язане одне важливе обмеження — на етапі компіляції всі обчислення повинні бути цілочисельними. Отже, ми не можемо ініціалізувати статичну константу типу `double`.

Зауважимо, однак, что наше настроювання стосувалося лишь обчислення середнього арифметичних. Для того щоб реалізувати шаблонну функцію, що обчислює середнє геометричне, довелося б практично цілком продублювати код, виправивши в ньому буквально три рядки: обчислення початкового значення лічильника, нагромадження добутку і повернення результату. Набагато практичніше було б написати функцію, що обчислює як середнє арифеметичне, так і середнє геометричне в залежності від шаблонного параметра. Зрозуміло, можна було б увести новий шаблонний параметр типу `bool`, але існує ще один спосіб, що дозволяє інтегрувати в одне целое как властивості, так і *поводження* класу. Цей спосіб заснований на понятті *стратегії*.

Опишемо особливості обчислення середнього арифметичного і середнього геометричних у виді окремого класу, що буде відігравати роль шаблонного параметра — класу стратегій.

## 17.2. Настроювання алгоритмів за допомогою стратегій

Слід одразу зауважити, що між класами характеристик та стратегій немає чіткої різниці. Стратегії мають багато спільного із характеристиками, але відрізняються від них тим, що в них менше уваги приділяється типам і більше поведінці.

> ***Клас стратегій*** *— це клас, що описує поведінку узагальнених функцій і типів.*

**Реалізація стратегій обчислення середніх значень**

```
#include <iostream>
using namespace std;

template<typename T>
class MeanTraits
{
public:
  typedef T MeanType;
  static MeanType start(T init)
     {  return init; }

};

template<>
class MeanTraits<char>
{
public:
  typedef int MeanType;
  static MeanType start(int init)
     {  return init; }
};

template<>
class MeanTraits<int>
{
public:
  typedef long MeanType;
```

```
    static MeanType start(long init)
      {  return init; }
 };

 template<>
 class MeanTraits<float>
 {
 public:
   typedef double MeanType;
   static MeanType start(double init)
     {  return init; }
 };

 class Arithmetic
 {
 public:
   template<typename T1>
   static T1 mean(T1* array, int N)
   {
      T1 result = MeanTraits<T1>::start(0);
      for(int i=0; i<N; i++)
      {
         result += *array;
         array++;
      }
      return result/N;
   }
};

 class Geometric
 {
 public:
   template<typename T1>
   static T1 mean(T1* array, int N)
   {
      T1 result = MeanTraits<T1>::start(1);
      for(int i=0; i<N; i++)
      {
         result *= *array;
         array++;
      }
      return exp(log(result)/N);
   }
};

 template <typename T,
           typename Policy = Arithmetic,
           typename Traits = MeanTraits<T> >
 class Mean
 {
 public:
   typedef typename Traits::MeanType MeanType;
   static MeanType mean(T * array, int N)
   {
      MeanType result = Policy::mean(array,N);
      return result;
   }
};

int main()
 {
```

```
    int array[] = { 1, 3, 5, 7, 9};
    cout << Mean<int>::mean(array,5) << endl;
    double vector[] = { 1, 9};
    cout << Mean<double,Geometric,MeanTraits<double> >::mean(vector,2);
    return 0;
}
```

Шаблонний клас `Mean` конкретизується трьома шаблонними параметрами: типом даних `T`, класом стратегії `Policy` і класом властивостей `Traits`. Тип даних `T` описує дані, що зберігаються в масиві. Клас `Policy` дозволяє реалізувати дві стратегії обчислення середніх значень. Якщо він заміняється класом `Arithmetic`, обчислюється среднеее арифметичне, а якщо класом `Geometric` — середнє геометричне. Клас `MeanTraits` призначений для опису властивостей результату і початкового значення.

Стратегії являють собою нове і дуже цікаву властивість мови C++. Повний опис їхніх особливостей і багато інших подробиць можна знайти в книзі Вандервурда і Джосаттіса.

## 17.3. Резюме

- Між характеристиками і властивостями немає чіткої межі.
- Характиристики більше уваги приділяють типам, а стратегії – поведінці.
- Характеристики можуть бути використані як фіксовані властивості, тобто без передачі їх шаблону як параметрів.
- Параметри характеристик, як правило, мають природні значення за замовчуванням, які дуже рідко перевизначаються або просто не можуть бути перевизначеними.
- Параметри характеристик мають тенденцію до сильної залежності від одного або кількох основних параметрів.
- Характеристики, як правило, містять типи і константи, а не функції-члени.
- Характеристики мають тенденцію до агрегації в шаблони властивостей.
- Класи стратегій практично завжди передаються як параметри шаблону.
- Параметри стратегій не обов'язково повинні мати значення за замовчуванням і часто спеціалізуються явно (хоча багато узагальнених компонентів налаштовуються із використанням стратегій, заданих за замовчуванням).
- Параметри стратегій, як правило, слабко пов'язані з іншими параметрами шаблону.
- Класи стратегій, як правило, об'єднують функції-члені в одне ціле.
- Стратегії можуть об'єднуватися у звичайних класах або шаблонних класах.
- Головна перевага використання стратегії за допомогою шаблонного параметру — спрощення ситуації, коли клас стратегії містить статичний член з типом, що залежить від параметрів шаблону.
- Головни недолік використання стратегій за допомогою шаблонного параметру полягає у тому, що класи стратегій повиння бути написані як шаблони із точним набором параметрів. Це виключає можливість додавати до класу стратегій додаткові параметри шаблону.

## 17.4. Контрольні питання

17.1. Чи є чітка межа між характеристиками і властивостями? Чому?
17.2. На які аспекти акцентовано увагу в класах характеристик і стратегій?
17.3. Як можуть бути використані характеристики?
17.4. Які параметри мають характеристики?
17.5. Яку тенденцію мають параметри характеристик?
17.6. Що містять класи характеристик на відміну від стратегій?
17.7. Які властивості мають характеристики?
17.8. Як передаються класи стратегій?
17.9. Чи обов'язково параметри стратегій повинні мати значення за замовчуванням?
17.10. Як пов'язані параметри стратегій з іншими параметрами шаблону?
17.11. Чии характеризуються класи стратегій?
17.12. Як об'єднуються стратегії?
17.13. У чому полягає головна перевага стратегій?
17.14. У чому полягає головний недолік стратегій?