Gem #6: Сравнение идиом различного представления в Ада 95 и интерфейсов Ада 2005

Автор: Matthew Heaney (On2 Technologies)

Краткое содержание: Gem Ада #6 – идиома различного представления позволяет осуществлять деривацию из множества помеченных типов одновременно. Этот механизм весьма удобен для составления абстракций, дополняющих типы интерфейсов.

Давайте начнем…

Одним из основных изменений в языке Ada является добавление интерфейсов, типов без учета состояния, которые определяют набор операций. Как и тегированный тип, вы можете получить тип интерфейса, но, в отличие от тега, вы можете выводить из нескольких интерфейсов одновременно.

Обычно вы используете тип интерфейса как своего рода спецификацию. Абстракция может быть записана в терминах типа интерфейса, что делает абстракцию полностью общей, поскольку она может использоваться с любым типом, реализующим этот интерфейс.

Ada95 не имеет типов интерфейсов, поэтому очевидный вопрос: можете ли вы создать эффект получения нескольких типов интерфейсов, но в Ada95? Ответ - да, используя технику, называемую идиомой «множественных просмотров». Эта методика позволяет типу предоставлять различные представления о себе, причем каждый вид имеет другой тип. Это очень похоже на наличие нескольких интерфейсов, но вам нужно самостоятельно строить инфраструктуру.

Чтобы проиллюстрировать разницу между интерфейсами и несколькими представлениями, мы сначала разработаем простую абстракцию для сохранения в терминах типа интерфейса, а затем перепроектируем ее с использованием техники множественных представлений. Интерфейс для типа Persistence_Type выглядел бы примерно так:

```
package Persistence_Types2 is
   type Persistence_Type is limited interface;

   procedure Write
     (Persistence : in Persistence_Type;
      Stream      : not null access Root_Stream_Type'Class) is abstract;
   … -- Read not shown here
end Persistence_Types2;
```

Любой тип, полученный из Persistence_Type, говорит о том, что он может быть сохранен на (и загружен с) с некоторой постоянной среды. Получение типа из интерфейса просто:

```
package P2 is
   type T is limited new Persistence_Type with null record;

   overriding
   procedure Write
     (Persistence : in T;
      Stream      : not null access Root_Stream_Type'Class);
   …
end P2;
```

Теперь все что приложение должно сделать, чтобы поддержать методы Persistence, это принять объект, который поддерживает интерфейс Persistence_Type. Вот наша абстракция для того, чтобы сделать это:

```
package Persistence_IO2 is
   … -- Initialization details omitted here
   procedure Save (Persistence : in Persistence_Type'Class);
end Persistence_IO2;
```

Это обеспечивает инфраструктуру, которая позволяет записывать любой тип (здесь, тип T), который происходит из Persistence_Type, на носитель. Реализация на основе файлов может выглядеть примерно так:

```
package body Persistence_IO2 is
   File : Ada.Streams.Stream_IO.File_Type;
   …
   procedure Save
     (Persistence : in Persistence_Type'Class)
   is
   begin
      Persistence.Write (Stream (File));
   end Save;
end Persistence_IO2;
```

Все что мы до этого делали, было с использованием интерфейсного типа Ada 2005, но мы ничего не делали такого, что строго требует типа интерфейса. Вы можете добиться такого же эффекта, используя тегированный тип, как это было бы сделано в Ada95. «Интерфейс» – это просто абстрактная помеченных нулем запись:

```
package Persistence_Types is
   type Persistence_Type is abstract tagged limited null record;

   procedure Write
     (Persistence : in Persistence_Type;
      Stream      : access Root_Stream_Type'Class) is abstract;
   … -- Read omitted here
end Persistence_Types;
```

Здесь Persistence_Type – это тегированный тип, а не тип интерфейса, но в остальном он такой же. Даже абстракция Persistence_IO такая же, как и раньше. Другое дело, как этот тип используется для поддержки сохранения в каком-то другом типе. Этот другой тип обеспечит «представление о сохранении». Мы хотим, чтобы тип поддерживал несколько представлений (так же, как это было бы при реализации нескольких интерфейсов), поэтому не просто получить прямой тип persistence, поскольку Ada не поддерживает вывод из более чем одного тега (это верно Как Ada95, так и Ada05).

Что мы сделаем для реализации представления persistence – создать промежуточный тип, который происходит из Persistence_Type, но с дискриминатором доступа, который обозначает родителя (тип, поддерживающий представление). Это позволяет операции промежуточного типа связываться с родительским типом, поскольку дискриминант обеспечивает доступ к состоянию родителя. Чтобы увидеть, как все это работает, давайте сначала покажем, как выглядит открытая часть родительского типа:

```
package P is
   type T is limited private;

   type Persistence_Class_Access is
     access all Persistence_Type'Class;
```

```
   function Persistence (Object : access T)
     return Persistence_Class_Access;
   … others views would be declared here
private
   … -- see text below
end P;
```

Тип T поддерживает сохранение, предоставляя функцию Persistence, которая возвращает значение доступа, обозначающее экземпляр Persistence_Type. «Persistence view» типа T получается путем вызова этой функции доступа. Этот механизм может быть расширен любым количеством раз, предоставляя несколько функций доступа, каждый из которых возвращает разные виды.

Функция Persistence возвращает значение доступа, которое фактически обозначает компонент записи T. Тип компонента – это промежуточный тип, который мы упоминали ранее, который происходит из Persistence_Type, объявленного следующим образом:

```
private
   type Persistence_View (Object : access T) is
     new Persistence_Type with null record;  -- no state req'd here

   procedure Write
     (Persistence : in Persistence_View;
      Stream      : access Root_Stream_Type'Class);
```

Обратите внимание, что тип Persistence_View – это нулевое расширение (оно не требует никакого собственного состояния), но с дискриминатором доступа, который обозначает родительский тип T. Это позволяет реализовать операцию Write, чтобы увидеть представление родительского типа, Поскольку параметр Persistence имеет дискриминант доступа, который обозначает экземпляр T, и поэтому операция имеет доступ ко всему состоянию, которое должно быть записано в persistent хранилище.

Родительский тип T реализуется при объявлении псевдонима компонент промежуточного типа:

```
   type T is limited record
      Persistence : aliased Persistence_View (T'Access);
      … -- rest of state here
   end record;
```

Компонент Persistence псевдоним, так как функция Persistence возвращает значение доступа, обозначающее этот компонент:

```
   function Persistence (Object : access T)
     return Persistence_Class_Access
   is
   begin
      return Object.Persistence'Access;
   end;
```

Другое различие между типами интерфейсов и идиомой множественных просмотров – это то, как клиент действительно вызывает операцию для выполнения операции сохранения. В случае интерфейса это просто: тип T происходит непосредственно из Persistence_Type, поэтому экземпляры типа T могут быть переданы в любую операцию, тип которой - Persistence_Type'Class. Пакет Persistence_IO имеет такую операцию, поэтому вызов будет выглядеть так:

```
procedure Test_Persistence2 is
   Object : T;
begin
   …
   Persistence_IO2.Save (Object);
end;
```

Вы должны сделать немного больше работы в случае с несколькими views, поскольку преобразование из T в Persistence_Type не является автоматическим. Здесь нам нужно явно вызвать функцию Persistence для «преобразования» из типа T в Persistence_Type'Class. Функция accessor имеет параметр доступа, поэтому мы должны объявить экземпляр типа T как aliased. Вызов выглядит следующим образом:

```
procedure Test_Persistence is
   Object : aliased T;
begin
   …
   Persistence_IO.Save (Persistence (Object'Access).all);
end;
```

Вот и все. Идиома многократных представлений – на самом деле очень мощный механизм для создания абстракций. Я охарактеризовал технику как идиому Ada-95, но заметьте, что даже в Ada-2005 это иногда полезно, например, когда вам нужно смешать тегированный тип с существующей иерархией. Наиболее распространенным примером является добавление контроля в тип листа в классе, потому что корневой тип сам по себе не вызывается из контролируемого.

Обсуждение…