Gem #48: Расширенные интерфейсы в Ada 2005

Автор: Quentin Ochem, AdaCore

Краткое содержание: Gem Ada #48 - В Ada 2005 представлена концепция интерфейсов, предназначенных для разработки классов объектов. Интерфейсы - удобный инструмент для реализации новых иерархии, но после введения их в действия их довольно трудно расширить. Добавление новых примитивов может привести к ошибкам механизма создания производных типов, так как типу необходимо реализовать все наследованные абстрактные примитивы. В данном Gem демонстрируются два способа преодоления данной проблемы, один из них - общий для ООП как такового, другой - специфичный для Ada 2005.

Давайте начнем…

Классический способ ООП

Предположим, у нас есть следующий интерфейс:

```
type Animal is interface;

procedure Eat (Beast : in out Animal) is abstract;
```

Все типы, реализовывая интерфейс Animal должны переопределить операцию Eat:

```
type Cat is new Animal with record ...

procedure Eat (Beast : in out Cat);
```

Теперь, спустя некоторое время, разработчик Animal может почувствовать необходимость позволить животным есть что-то конкретное, и хотел бы добавить следующую операцию в интерфейс:

```
procedure Eat (Beast : in out Animal; Thing : in out A_Thing);
```

К сожалению, существуют сотни видов животных, реализующих этот интерфейс, и необходимость мигрировать всем под новый интерфейс будет слишком болезненной. Не говоря уже о том, что большинство из них даже не нуждаются в этом новом способе питания, то есть в новом интерфейсе – они просто счастливы есть какое-то случайное количество анонимной пищи. Просто расширение этого интерфейса не путь решения задачи - так что расширение должно быть сделано отдельно, в новом интерфейсе, например:

```
type Animal_Extension_1 is interface;

procedure Eat (Beast : in out Animal_Extension_1; Thing : in out A_Thing) is abstract;
```

Итак, теперь нужно объявить животных, которые должны полагаться на этот новый способ еды, например:

```
type Cat is new Animal and Animal_Extension_1 with record ...
```

Обратите внимание, что можно даже обеспечить тот факт, что расширение животного (Animal_Extension_1) должно быть в первую очередь животным, написав:

```
type Animal_Extension_1 is interface and Animal;
```

что приведёт к более простому объявлению для типа Cat, поскольку больше нет необходимости расширяться от двух интерфейсов:

```
type Cat is new Animal_Extension_1 with record ...
```

В результате этого изменения остальная часть кода останется полностью нетронутой. Вызов новой подпрограммы потребует некоторого дополнительного объёма работы, поскольку мы сначала должны проверить, что тип Animal, с которым мы имеем дело, действительно является потомком Animal_Extension_1 и выполняет преобразование в класс этого интерфейса, перед вызовом новой версии Eat:

```
The_Animal : Animal'Class := ...

if The_Animal in Animal_Extension_1'Class then
   Animal_Extension_1'Class (The_Animal).Eat (Something);
end if;
```

Способ доступный в Ada 2005

Язык программирования Ada 2005 вводит понятие нулевых процедур. Процедура null-это процедура, объявленная с помощью "is null" и логически имеющая пустое тело. К счастью, нулевые процедуры разрешены в определениях интерфейса - они определяют поведение по умолчанию такой подпрограммы как ничего не делать. Возвращаясь к примеру Animal, программист может объявить примитив интерфейса Eat следующим образом:

```
procedure Eat (Beast : in out Animal; Thing : in out A_Thing) is null;
```

Все наши сотни видов животных автоматически наследуются из этой процедуры, но не должны будут реализовывать её. Добавление этого объявления не повреждает исходную совместимость с контрактом интерфейса Animal. Кроме того, поскольку никакие новые типы не включены, намного проще выполнить вызовы к этой подпрограмме - больше нет необходимости проверять членство и производить преобразование типов, и мы можем просто записать:

```
The_Animal : Animal'Class := ...

The_Animal.Eat (Something);
```

подпрограмма будет выполняться обычным образом с учётом исключения `The_Animal`, которых явно переопределили.

Связанный исходный

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Quentin Ochem имеет опыт разработки программного обеспечения, специализируется на разработке программного обеспечения для критически важных приложений. Он имеет более чем 10-летний опыт в разработке Ada. Сегодня он работает техническим менеджером AdaCore по проектам, связанным с авионикой, железной дорогой, космосом и оборонной промышленностью. Он также преподает курс авионики стандарта DO-178B в университете EPITA в Париже.

Last Updated: 10/13/2017
Posted on: 10/13/2008

Обсуждение…