IMPLEMENTACIÓN DE LA HERENCIA EN ADA

RESUMEN
La herencia es uno de los paradigmas más poderosos con que cuenta la Programación Orientada a Objetos, para facilitar el desarrollo de software, sobre todo cuando se trata de grandes sistemas, en donde este mecanismo es fundamental para escribir programas cortos, compactos y sobretodo sin código redundante.

ADA, aunque no es un lenguaje especialmente orientado Orientado a Objetos, ofrece mecanismos para el desarrollo bajo esta metodología.

Este artículo pretende exponer algunas ideas sobre como implementar la herencia en ADA, a través de tres de sus recursos: los Registros Variantes, el concepto de Paquetes – Packages – y los Genéricos – Generic -.

PALABRAS CLAVES: Registros Variantes, Paquetes y Genéricos

JORGE IVAN RIOS PATIÑO
Profesor Ingeniería Sistemas y Computación
Universidad Tecnológica de Pereira
jirios@utp.edu.co

ABSTRACT
The inheritance, is one of the more powerful paradigms the Oriented Objects Programming have it, to make the software development easier, specially big systems, where this mechanism is fundamental to write short and compact programs without redundant code.

ADA, although isn't a language specially oriented for the Oriented Objects, but offers mechanisms for the software development under this methodology.

This article, pretends expose some ideas about how to implement the inheritance in ADA, with three of it's resources: Variants Records, the Packages concept and the Generics.

KEYWORDS: *Variants Records, Packages and Generics.*

1. INTRODUCCIÓN

Cuando se desarrolla software, sobretodo grandes sistemas, uno de los errores mas frecuentes que se cometen, es la redundancia de código y la no reducción de especificaciones, que hacen que dichos programas sean inútilmente voluminosos y difíciles de mantener.

La Programación Orientada a Objetos – POO - de aquí en adelante, provee a través de la *herencia*, un mecanismo para disminuir la redundancia y simplificar la especificación de aquellos componentes de software que son similares a otros. Para esto emplea el concepto de *Clase*, que básicamente es una abstracción para definir un conjunto o familia de objetos o subclases, de un sistema, que comparten sino la totalidad, una buena parte de sus características.

Este mecanismo, como otros tantos de la POO, solo viene implementado explícitamente en aquellos lenguajes que claramente son orientados hacia este nuevo paradigma, como JAVA, C++, SMALLTALK, MOZART, EIFFEL, CLOS, entre muchos otros. En los lenguaje que no soportan la POO, como C, FORTRAN, PASCAL, etc. esta característica tiene que simularse. Aunque ADA no se clasifica estrictamente en el segundo grupo, dado que ofrece otras características de tipo hereditario, como *new* para instanciaciones, discriminación de registros y el encapsulamiento de datos a través de los paquetes – Packages -, la generalización de tipos – Generic -, es necesario emplear ciertas abstracciones para implementar la *herencia*.

Este artículo, establece como se puede implementar la *herencia* en ADA, a través de los mecanismos antes enunciados.

2. EL PARADIGMA DE LA HERENCIA

Unos de los mecanismos fundamentales de la POO, es el paradigma de la *herencia*, que en su concepto mas general lo que hace es modelar un sistema por medio de objetos que representan sus clases, con lo cual se conforman jerarquías y cuando se relacionen esas *clases* con otras, se conforma toda una gran red que se puede tomar como el modelo que representa el sistema en cuestión.

Fecha de Recibo: 21 abril de 2003
Fecha de Aceptación: 20 de mayo de 2003

Esto, ayuda a diseñadores y programadores a escribir programas cortos, compactos, no redundantes y sobretodo capaces de recibir nuevas especificaciones sin que ello conlleve cambios en las estructuras de los mismos.

Por ejemplo, en una Universidad, se pueden identificar Profesores, Estudiantes y Empleados (ver Fig. No. 1). Si miramos sus características mas relevantes – nombres, apellidos, cédula, fecha de nacimiento, etc. -, estos pueden ser englobados bajo el concepto de *Persona*, dado que en esta *clase* abstracta, se pueden resumir y representar la mayoría de las características antes mencionadas.

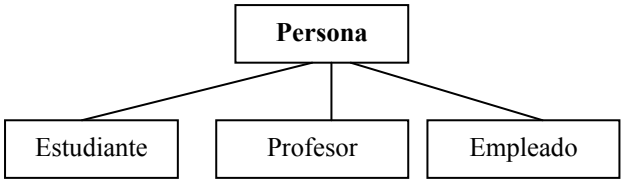

Figura 1. Ejemplo de Clases

Cuando se necesita definir una subclase, por ejemplo – *Estudiante*- de *Persona*, lo que hace es generar una instancia de la *clase* superior que lo engloba, heredando todas características y demás ítems, adicionando las nuevas componentes diferenciales del nuevo objeto, con lo cual se obtiene un desarrollo incremental del software.

Además, de manera recursiva, si se necesita de esta nueva *clase* instanciada -*Estudiante*- generar otra subclase, por ejemplo *Estudiante Asistente* (Ver Fig. No. 2), se genera de nuevo una instancia de *Estudiante*, con lo cual la nueva subclase, hereda todo lo de su *padre*, que incluye lo que a su vez el heredó de *Persona*, y los ítems agregados cuando se creó.

Figura 2. Ejemplo de Instancias.

Parte fundamental de la POO y de la *herencia,* tiene que ver el encapsulamiento (ver Fig. No. 3) de las características de las *clases*, que permite que, un programador manipule objetos sin que tenga que conocer los detalles de su representación interna, lo que impide manipular las estructuras internas, manteniendo el control del desarrollo y de las especificaciones de los programas..

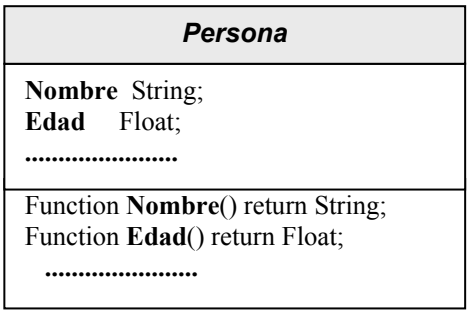

Figura 3. Ejemplo de Encapsulamiento

3. EL LENGUAJE ADA

Este lenguaje, creado a instancias del Departamento de Defensa de los Estados Unidos, hacia finales la década de los 70', es un intento por obtener un lenguaje que remplazara los cerca de 1.300 lenguajes de programación que esa institución usaba en ese entonces.

Fuertemente tipado – comprobación exhaustiva de tipos -, emplea mecanismos de herencia implícitos, utilizando registros variantes, el concepto de modularización de programas bajo el método de paquetes, y la generalización de tipos.

Según G. Booch [1] ADA ofrece las siguientes características, para la POO:

- Clases de tipos de datos abstractos pueden ser implementados por unidades genéricas.
- Clases de Objetos pueden ser implementados por paquetes que exportan tipos Privados – Private – o Limitadamente Privados – Limited Private -.
- Objetos son implementados por instancias de tipos Privados o Limitadamente Privados o como paquetes que sirven como máquinas de estados.
- Operaciones – métodos -, son implementados como subprogramas exportados de las especificaciones de un paquete; los parámetros generales formales de los procedimientos sirven para especificar las operaciones requeridas de un Objeto.
- Variables sirven como nombres de Objetos; alias son permitidos.
- La visibilidad es estáticamente definida a través de cláusulas de unidades de contexto.
- Compilación separada de la especificación de paquetes y sus cuerpos, soportan dos vistas de un mismo objeto.
- Tareas y tipos de tareas pueden ser usadas para implementar objetos y clases de objetos actores.

4. IMPLEMENTACION DE LA HERENCIA EN ADA

- **Registros Variantes**

En ADA, el tipo registro – Record – puede ser definido de la siguiente forma (ver Fig.No. 4):

```
Type
Record Nombre (Parámetros) Is
   Campo 1 : Tipo 1 [:= ValorInicial 1];
   Campo 2 : Tipo 2 [:= ValorInicial 2];
          .....
   Campo n : Tipo n [:= ValorInicial n];
   [Case Parametro Is
      When Opcion 1=>Campo j:Tipo j [:= Val.Inic. j];
      When Opcion 2=>Campo k:Tipo k [:= Val.Inic. k];
      .....
      When Opcion 2=>Campo y:Tipo x [:= Val.Inic. x];
    End]
End
```

Figura 4. Estructura general de Registro Variante

Los discriminantes, en la definición del tipo registro – Record - , actúan de la misma forma como parámetros de las funciones y procedimientos en ADA.

Siguiendo con el ejemplo ya enunciado, una *Clase* se implementaría de la siguiente forma (ver Figura No. 5):

```
Type SubClase (Estudiante, Profesor, Empleado);
Type Genero (Masculino, Femenino);
Type Sexo(Hombre, Mujer);
Type EstadoCivil (Soltero, Casado, Viudo, UnionLibre);
Type TipoEmpleado(Oficial, Publico);
Type Real Is New Float;
SubType RangoSalario Is Real 644000..1000000;
Type  Record Fecha Is
   Dia : Short_Integer Range 1..31;
   Mes : Short_Integer Range 1..12;
   Año : Short_Integer Range 1930..2090;
End Record;
Type
Record Persona (Clase : SubClase, Tipo: Genero) Is
   Nombre    : String(1..30);
   Apellidos  : String(1..30);
   Cedula     : String(1..12);
   FechaNacimiento : Fecha;
   FechaIngreso : Fecha;
   Estado : EstadoCivil;
   Case Genero Is
      When Masculino => Sexo := Hombre;
      When Femenino => Sexo := Mujer;
   End Case;
   Case SubClase Is
      When Estudiante =>
         Facultad : String(1..15);
         Semestre : Short_Integer Range 01..10;
         PromedioCarrera : Real Range 0.0..5.0;
      When Profesor =>
         Categoria : Short_Integer Range 01..10;
         Area: String(1..15);
         Salario : RangoSalarios;
      When Empleado =>
         Clasificacion : TipoEmpleado;
         Cargo : String(1..15);
   End Case;
End Record;
```

Figura 5. Ejemplo de definición de una *Clase*

Esta estructura define, implícitamente, la *clase* llamada *Persona*, pero por medio de los discriminantes – *SubClase* y *Género* – podemos crear otras subclases de la siguiente forma (Ver Figura No. 6):

```
Type Estudiante Is New Persona(Clase => Estudiante);
```

Figura 6. Ejemplo de creación de la *SubClase* Estudiante

Automáticamente ADA genera una instancia –*subclase*- denominada *Estudiante*, que hereda todos los atributos de la *clase* que lo generó, en este caso *Persona*.

Nótese que solamente se ha utilizado uno de los dos parámetros definidos en *Persona*, con el fin de que si requiere una nueva subclase de *Estudiante*, por ejemplo, crear las subclases de Estudiante con diferencia de género, se puedan generar asi: (Ver Figura No. 7):

```
Type Estudiante_Maculino Is New
                    Estudiante (Genero => Masculino);
```

Figura 7. Ejemplo de creación de una nueva *Subclase*

O aún puede haberse generado en un solo paso (Ver Figura No. 8):

```
Type Estudiante_Maculino Is New
                    Estudiante (Estudiante, Masculino);
```

Figura 8. Creación de la anterior Subclase en un paso

- **Paquetes – *Packages* –**

Nótese que mediante el concepto de registro variante, solo hemos podido definir la estructura de la *clase*, en este caso *Persona*, mas no sus *métodos*. En Ada no es posible como en C, tener componentes en su estructura que sean apuntadores a funciones, con lo cual podríamos definir los métodos del objeto. En ADA, esto es posible mediante el concepto de Paquetes, que aparte de descomponer física y lógicamente los programas,

mediante ellos, podemos agrupar todo lo referente a una *clase*.

En su forma mas general, un paquete en ADA puede ser de la siguiente forma (Ver Figura No. 9):

```
Package Nombre Is                          } Parte Visible
        Declaraciones                      }
End Nombre;

Package Body Is Nombre Is                  }
        Definiciones                       }
                                           } Parte No Visible
Begin                                      }
        Inicializaciones                   }
End Nombre;
```

Figura 9. Estructura general de un Paquete

Mediante la implementación de los paquetes, podemos encapsular todo lo referente a la *clase* que vamos a definir.

Siguiendo con el ejemplo, tenemos (Ver Figura No. 10):

```
Package Persona Is
   Function Nombre( P : Persona) Return String;
   Function Apellidos( P : Persona) Return String;
    .....
   Procedure P_Nombre(P:Persona; Nombre:String);
   Procedure P_Apellido(P:Persona; Apellido:String);
    .....
     demás métodos de la clase Persona
End Persona;
```

Figura 10. Uso de Paquetes para definir una clase

La segunda parte del paquete *Persona* y a la vez la no visible del mismo se define así (Ver Figura No. 11):

```
Package Body ClasePersona Is
    Type SubClase (Estudiante, Profesor, Empleado);
    Type Genero (Masculino, Femenino);
    .....
    demás declaraciones de tipos y constantes
    Type
    Record Persona(Clase:SubClase;Tipo:Genero) Is
        Nombre    : String(1..30);
        Apellidos : String(1..30);
        Cedula    : String(1..12);
        .....
        Case Genero Is
            When Masculino => Sexo := Hombre;
        .....
        resto de las declaraciones de Persona
    End Persona;

    Function Nombre( P : Persona) Return String Is
    Begin
        .....
        definición del cuerpo de la función Nombre
        .....
    End Nombre;

    Function Apellidos( P : Persona) Return String Is;
    Begin
        .....
        definición del cuerpo de la función Apellidos
        .....
    End Nombre;
    Procedure P_Nombre(P: In Out Persona;
                       Nombre: In String) Is;
    Begin
        .....
        definición del cuerpo del procedimiento
         P_Nombre
        .....
    End P_Nombre;
        .....
        demás métodos de la clase ClasePersona
        .....
End ClasePersona;
```

Figura 11. Cuerpo del Paquete Persona

- **Herencia**

Consideremos una *clase* llamada *Estudiante*, que comparte una buena parte de la información de la *clase* superior que lo genera; para esto utilizamos el mecanismo de instanciación – New – con el paquete *Clase Persona* (Ver Figura No. 12).

```
With ClasePersona; Use ClasePersona;
Package ClaseEstudiante Is
   Type Estudiante Is New
                Persona (SubClase => Estudiante);
End ClaseEstudiante;
```

Figura 12. Creación de *Clases* mediante Paquetes

El paquete *ClaseEstudiante* lo que hace es simular esta nueva *subclase*, heredando todo los componentes – estructuras y métodos – definidos en el *padre*.

Un punto importante es que esta nueva *clase* – *ClaseEstudiante* - podemos agregarle nuevos métodos, o métodos propios a la misma. Por ejemplo, un método específico de *ClaseEstudiante*: podría ser el

procedimiento *Matrícula*, el cual se agregaría de la siguiente forma (Ver Figura No. 13)

```
With ClasePersona; Use ClasePersona;
Package ClaseEstudiante Is
   Type Estudiante Is New
                    Persona (SubClase => Estudiante);
    Procedure Matricula(E : In Out Estudiante);
     .....
    Demás métodos nuevos de la clase Estudiante
    .....
End ClaseEstudiante;

Package Body ClaseEstudiante Is
  Procedure Matricula(E : In Estudiante) Is
  Begin
  .....
  End;
   .....
    Demás métodos nuevos de la clase Estudiante
    .....
End ClaseEstudiante;
```

Figura 13. Forma de agregar nuevos métodos

De esta forma generamos la *clase Estudiante*, la cual hereda de la estructura *Persona*, los atributos comunes, y debido a la utilización del discriminante *Subclase=>Estudiante*, se heredan los atributos propias del mismo.

Si aún quisiéremos crear otra subclase de esta última instanciada, por ejemplo distinguiendo el género, se haría de la siguiente forma y de manera recursiva como en el ejemplo anterior (Ver Figura No. 14):

```
With ClaseEstudiante; Use ClaseEstudiante;
Package EstudianteMasculino Is
   Type EstudianteMasculino Is New
                    Estudiante (Genero => Masculino);
 .....
 Demás métodos nuevos de la clase EstudianteMasculino
 .....
End EstudianteMasculino;

Package Body EstudianteMasculino Is
.....
 Demás métodos nuevos de la clase EstudianteMasculino
 .....
End EstudianteMasculino;
```

Figura 14. Instancias mediante Paquetes

- **Genéricos**

Nótese que hasta ahora se han tenido que definir exhaustivamente los tipos que se utilizaron en la definición de las *clases*, con lo cual la instanciación de los nuevos ejemplares, queda circunscrita a los tipos ya predefinidos. La creación de una nueva *clase* que no esté definida en los tipos, generaría la inclusión del mismo en la definición y por consiguiente una nueva compilación de los módulos – Packages – con la problemática que esto generaría. Por ejemplo: si identificáramos *Obreros* como una nueva subclase, no contemplada en la fase inicial del diseño, tendríamos que incluir esta, de nuevo, en las definiciones de los tipos.
Además se puede generar una duplicación de código o polimorfismo, en algunos métodos de las *clases* superiores, dado que en muchas ocasiones la única variación en los mismos es el tipo de parámetros.

El mecanismo Genérico – Generic – ayuda a solucionar este problema, debido a que mediante él, podemos definir tipos y procedimientos como parámetros. Adicionalmente, se pueden implementar la visibilidad con dichos tipos, utilizando dos restricciones para los mismos: Private y Limited Private;
Por lo tanto, en ADA, es posible tener paquetes genéricos, lo que en principio sirve para hacer abstracciones en la definición e instanciación de *clases* en POO.

Siguiendo con el ejemplo que traíamos, la definición de la C*lase Persona* sería así (Ver Figura No. 15):

```
Generic
  Type SubClase Is Private;
  Type EstadoCivil Is Private;
  Type Genero Is Private;

Package ClasePersona Is
   Type Persona(Clase:SubClase;Tipo:Genero)
                                  Is Limited Private;
   Function Nombre( P:Persona) Return String;
   Function Apellidos(P:Persona) Return String;
    .....
   Procedure P_Nombre(P: Persona In Out;
                      Nombre: In  String);
   Procedure P_Apellido(P: Persona In Out;
                      Apellido: In String);
     .....
     demás métodos de la clase Persona
     .....
   Private
   Type
     Record Persona(Clase:SubClase;Tipo:Genero) Is
          Nombre    : String(1..30);
          Apellidos  : String(1..30);
          Cedula     : String(1..12);
          FechaNacimiento : Fecha;
          FechaIngreso : Fecha;
          Estado : EstadoCivil;
          Case Genero Is
                  When Masculino => Sexo := Hombre;
          .....
          resto de las declaraciones del registro Persona
          End;
       End Persona;
End ClasePersona;
```

Figura 15. Uso de Genéricos

El cuerpo del paquete – Package Body - se define de la misma forma que hemos mostrado antes.

El código anterior, crea un paquete genérico que corresponde de manera abstracta a la *clase Persona*, sin tipos previamente definidos. Instanciar una nueva clase, por ejemplo *Obrero* se haría en el momento en que se definiera sus tipo (Ver Figura No. 16).

```
.....
Type SubClase (Estudiante, Profesor, Empleado, Obrero);
Type Genero (Masculino, Femenino);
Type Sexo(Hombre, Mujer);
Type EstadoCivil (Soltero, Casado, Viudo, UnionLibre, Otro);
Type TipoEmpleado(Oficial, Publico, Mixto);
.....
Package ClaseObrero Is New
                ClasePersona (SubClase =>Obrero;);
```

Figura 16. Creación de subclases mediante Genéricos

De manera recursiva, podríamos definir nuevas clases de la anteriormente instanciada, por ejemplo, definiendo el Genero (Ver Figura No. 17).

```
Package ClaseObrera Is New
                Obrero (Tipo => Femenino);
```

Figura 17. Creación de una nueva *Subclase*

Esto en un solo paso (Ver Figura No. 18).

```
Package ClaseObrera Is New
   ClaseObrero (SubClase =>Obrero;Tipo=> Femenino);
```

Figura 18. Creación de una nueva *Subclase* en un paso

Esto visto gráficamente es así (Ver Figura No. 19).:

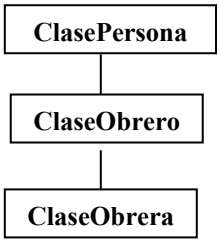

Figura 18. Representación gráfica del ejemplo anterior

Los genéricos, actúan como parámetros de los paquetes, instanciando lo que se quiere de la clase superior.

Este tipo de instanciaciones, requiere un cierto grado de disciplina a la hora de utilizar los paquetes generados. En ADA toda instanciación, hace que se herede todo lo concerniente al padre que lo deriva. El uso incorrecto de estos recursos heredados, pueden hacer mas denso el desarrollo del sistema.

5. CONCLUSIONES

- A pesar de que ADA no es un lenguaje explícitamente orientado hacia la POO, como se ha demostrado, pueden implementarse dichas características, sobre todo la *herencia*, mediante el adecuado manejo de sus registros variantes, los paquetes genéricos y una adecuada disciplina de programación.

- La carencia explícita de los mecanismos de la POO, en ADA, ya se han venido solucionando, por ejemplo, con la extensión misma del lenguaje a través de ADA++ [2] o de la creación de DRAGOON (An ADA Object Oriented Language) [3]

- Según J. Rambaugh [4] , la potencia de un lenguaje, se mide por la expresividad y la facilidad de la implementación de las abstracciones necesarias, que se necesitan a la hora de realizar implementaciones, que de su poder de computación.

6. BIBLIOGRAFÍA

6.1 Bibliografía Referenciada

[1] BOOCH, Grady, "Software Components in ADA: Structures, Tools and Subsystems", Benjamin Cumings, Menlo Park, California, 1.988

[2] FORESTIER, J.P, FORNARINO, C., FRANCHI-ZANNETACCI, P., "ADA ++, A Clas and Inheritance Extensions for ADA", The ADA Companioon Series, ADA: the design choice, Proceedings of the Ada-Europe, International Conference, Madrid, España, 1.989

[3] DI MAIO, Andrea y otros., "DRAGOON: An ADA-based Object Oriented Language", The ADA Companioon Series, ADA: the design choice, Proceedings of the Ada-Europe, International Conference, Madrid, España, 1.989

[4] RAMBAUGH, James y otros, "Modelado y Diseño Orientado a Objetos, Metodología OMT", Ed. Prentice-Hall, Madrid, 1.995

6.2 Bibliografía Consultada

[1] J.P.G. Barnes, "Programación en ADA", Ediciones Diaz de Santos, S.A, Madrid, 1.987

[2] BOOCH, Grady, RAMBAUGH, James y JACOBSON, Ivar, "El Lenguaje Unificado de Modelado: UML, "Editorial Addison esley:, Madrid , España, 2000.