Un Estudio de Aspectos de Lenguaje de Programación de Bases de Datos Disponibles en PJama

Gustavo Larriera [glarrier@ei.edu.uy]
Univ. Autónomo del Sur - Lab. Sistemas de Información [http://www.silab.ei.edu.uy/]
Agosto 1998.

Resumen. En este trabajo se estudian las características del sistema de tipos persistente de PJama, una implementación en desarrollo del lenguaje Java. El sistema PJama provee mecanismos de persistencia al lenguaje Java sin cambios en el lenguaje, que pueden revisarse desde el punto de vista de las bases de datos orientadas a objetos. El estudio realizado analiza las siguientes características: Naturaleza del sistema de tipos de PJama, su expresividad, propiedades de tipos y valores, y relaciones existentes entre los tipos. También se revisan aspectos típicos de los sistemas de bases de datos y las posibilidades que puede tener PJama al respecto.

***Palabras clave**: Lenguaje persistente; Java; orientación a objetos; sistema de tipos; lenguaje de base de datos; polimorfismo; herencia; transacción.*

Contenido

---

1 Introducción

El proyecto *PJama* es un trabajo en colaboración entre los laboratorios de Sun Microsystems y el Departamento de Computación de la Universidad de Glasgow, cuyo objetivo es dotar al lenguaje de programación Java de un sistema de tipos persistente, sin tener que modificar al lenguaje [AJD*96]. El resultado es un lenguaje denominado PJama, originalmente denominado PJava. PJama dispone de clases adicionales que proveen la persistencia a través de una arquitectura que permite al alcenamiento y manejo de los objetos persistentes [PAD*97]. El resto de las clases de Java se utilizan sin modificaciones visibles al usuario.

En este artículo se realiza un estudio de las características disponibles en el sistema de tipos persistente de PJama. A los efectos de disponer de un marco de referencia para el estudio, se abordan los siguientes aspectos tal como se describen en [ADG*89]: Naturaleza del sistema de tipos, expresividad, propiedades de tipos y valores, y relaciones existentes entre los tipos yclases. Adicionalmente se analizan algunas cuestiones propias de los sistemas de bases de datos, como el manejo de transacciones, recuperación frente a errores y seguridad.

El resto del artículo se divide en las siguientes secciones: La sección 2 presenta las principales características de PJama; la sección 3 está dedicada a la naturaleza del sistema de tipos de PJama; la sección 4 está dedicada

a describir la expresividad del lenguaje (tipos primitivos disponibles, constructoras de tipos nuevos y estilo de polimorfismo); la sección 5 está dedicada a los tipos y valores del sistema (propiedades, semántica de la igualdad); la sección 6 está dedicada al estudio de las propiedades de las relaciones entre los tipos (equivalencia, mecanismos de herencia); la sección 7 está dedicada a las nociones de tipos y clases, manipulación de las mismas y de sus objetos; la sección 8 considera cuestiones de interés para la comunidad de las bases de datos. Finalmente la sección 9 plantea un resumen y conclusiones.

2 El lenguaje PJama

El lenguaje PJama es una implementación, actualmente en desarrollo, que agrega persistencia de propósito general al lenguaje Java, con un mínimo de perturbación a su semántica e implementación original como primer meta a lograr. Otras metas seguidas en el diseño de PJama consisten en lograr la persistencia de los objetos en forma segura, con mínimo impacto en la performance, cambios mínimos en los aspectos del lenguaje Java y modelo de transacciones [AJD*96].

Los principios de diseño fundamentales son [ADJ*96]: (a) *Persistencia ortogonal*, es decir que cualquier objeto de cualquier clase puede persistir incluyendo a los tipos de datos primitivos; (b) *Persistencia transitiva,* que establece que durante la ejecucion normal de un programa PJama, los objetos persisten en tanto sean alcanzables desde alguna variable; y (c) *Independencia de la persistencia,* donde se requiere que sea indistinguible cuándo el programa está operando sobre datos persistentes o no. A los efectos de visualizar el aspecto de un programa PJama, podemos usar el siguiente ejemplo (extraido y adaptado de [ADJ*96):

```
public class SaveSpag {
   public static void main (String[] args) { // comienza transaccion
      Spaghetti sp1 = new Spaghetti(27); // crear objeto nuevo
      Spaghetti sp2 = new Spaghetti(5); // crear otro objeto
      sp1.add("pesto"); // modificar un objeto
      sp1.add("pimienta");
      sp2.add("queso");
      try { // capturar excepciones de almacenamiento
         PJavaStore pjs = PJavaStore.getStore(); // persistencia
         pjs.newPRoot("Spag1", sp1); // hacer persistente
      } catch (PJSException e) { ... } // manejo de excepciones
   } // end main, commit
} // end SaveSpag
```

PJama agrega a la API disponible para el usuario una clase PJavaStore que provee acceso a las funcionalidades del mecanismo de persistencia. En el ejemplo, el método newPRoot asocia al string "Spag1" con el objeto *sp1*, y lo almacena en forma persistente. Al terminar el programa *main*, automaticamente se produce un COMMIT que provoca que los datos identificados como persistentes y todos los objetos alcanzables transitivamente desde ellos, sean promovidos al estado "persistente" (e.g. *sp1*). En contrapartida, las estructuras de datos asociadas a *sp2* no persisten pues no son alcanzadas por objeto persistente alguno. En el ejemplo, se convierten en persistentes: el objeto *Spaghetti Class*, todas las clases usadas para definirlo y otros objetos que sean referenciados desde él.

3 Naturaleza del Sistema de Tipos

El sistema de tipos de PJama coincide con el sistema de tipos del lenguaje Java. Los diseñadores de PJama decidieron agregar persistencia a Java con un mínimo de perturbación a la semántica e implementación original de Java [AJD*96]. Bajo esas circunstancias tenemos entonces que PJama es un lenguaje fuertemente tipado, con chequeo estático. El sistema de tipos de Java está bien definido y es rigurosamente verificado por el compilador. Los métodos deben declararse explícitamente (no se permiten declaraciones implícitas al estilo de C). La validación de los tipos usados en las expresiones se realizan durante compilación, aún cuando la ejecución del programa es interpretada. Los detalles de implementación de los tipos (e.g. formato interno de

almacenamiento) están ocultos a nivel del programa, aunque el usuario en cierta forma puede elegir el tamaño usado para almacenar determinados valores, por ejemplo al elegir alguno de los tipos numéricos.

4 Expresividad

4.1 Tipos primitivos y constructores de tipos

Los tipos básicos (primitivos) disponibles en PJama son los mismos de Java. Es decir, son similares a los disponibles en C con la excepción de que los valores booleanos están bien diferenciados (tipo `boolean`) y de que no se dispone de tipos puntero. Las constructoras disponibles son `class`, `interface` y array (que se denota con [ ]). La constructora `class` implementa herencia simple y permite definir un conjunto de atributos de clase, con posibilidad de combinar propiedades de visibilidad (`public` , `protected` , `private`), atributos de clase (static) o de instancias (`dinamic`). En la tabla siguiente se presentan los constructores de tipos habituales en los lenguajes de programacion, tal como se describen en [CW85] y se señalan cuáles están disponibles en PJama/Java:

CONSTRUCTOR	EN PJama/Java	COMENTARIOS
Tipos primitivos	byte, short, int, long, float, double, boolean, char	• Los tipos primitivos no están implementados en clases, aunque existen clases apropiadas denominadas "type wrappers" para cada tipo primitivo. • Se permite "cast" entre tipos numéricos, aunque solamente se garantiza que no haya pérdida de información en los casos donde el tipo receptor sea de mayor o igual tamaño de almacenamiento que el el tipo origen. • No se dispone de tipo String en forma primitiva, aunque existe una clase a tales efectos. • No se dispone de tipo apuntador. • El valor default para cualquier variable de un tipo primitivo es null. • Son pasados por valor. Los tipos no primitivos (arrays y objetos) son pasados por referencia.
Producto (par, tupla)	NO	
Registro	SI	• No existe una constructora explícita, todas las variables declaradas en la interfaz de una clase actúan como atributos de registro. • Las operaciones selectoras se crean automáticamente con el nombre *getNombre*, donde *Nombre* es el nombre de un atributo del registro. Las selectoras automáticas tienen visibilidad desde otras clases del mismo package (grupos de clases). • Se pueden especificar otras operaciones selectoras mediante public, protected y private.
Enumerados	NO	• En parte pueden simularse con el uso de constantes static final.
Unión discriminada	NO	• Se puede simular usando subclases.
Registro variante	NO	• Se puede simular usando subclases.
Array	SI	• Se declaran usando la forma *Tipo NomArray[]* y su dimensión se

		asigna al ser creados usando new. • Se permiten arrays multidimensionales. • Siempre son pasados por referencia.
Lista	NO	
Conjunto	NO	
Secuencia	NO	
Función	SI	• Métodos en las clases.
Referencia (apuntador)	NO	
Valor mutable	NO	
Subtipo mediante predicado	NO	
Tipo paramétrico	NO	
Tipo abstracto de datos	SI	• Se simulan usando clases y declaraciones de visibilidad en las interfaces de las clases.
Definición recursiva de tipos	SI	

4.2 Polimorfismo

La noción de tipo/subtipo en PJama solamente es aplicable al concepto de clase/subclase. Los únicos tipos disponibles son los tipos primitivos y no tienen definida una jerarquía subtipo. Todos los objetos deben pertenecer a alguna clase y la jerarquía entre clases se establece con la declaración extends. PJama dispone de los siguientes tipos de polimorfismo:

- *Polimorfismo por inclusión:* Cuando una clase C es subclase de una clase C', entonces cualquier método aplicable a un objeto de clase C' puede aplicarse a un objeto de clase C.
- *Polimorfismo paramétrico explícito:* La clase efectiva de un parámetro y el resultado de un método deben especificarse explícitamente.
- *Polimorfismo ad hoc (overloading):* Un método puede actuar en forma diferente dependiendo de la clase del objeto al que se le aplica.

5 Tipos y Valores

5.1 Propiedades de tipos y valores

En PJama, todos los valores deben pertenecer a un tipo primitivo y todos los objetos deben pertenecer a alguna clase. Los objetos persistentes sintácticamente no se diferencian de los que no lo son, aunque explícitamente se consigue que un objeto sea persistente almacenándolo en un "store". El estado interno de cada objeto es independiente de su identidad: todos los objetos creados son diferentes entre sí aunque tengan los mismos valores en su estado interno. Debido a que todos los objetos y arrays son pasados por referencia en forma automática, dos variables diferentes pueden referenciar al mismo objeto. De esta forma, asignar un objeto a otro, no realiza una copia del valor sino que simplemente se asigna una referencia. Para copiar los datos de un objeto se puede utilizar el método clone(), y en el caso de quererse copiar un array entero puede utilizarse el método arraycopy(). Lo anterior no ocurre con los tipos primitivos. Por ejemplo:

```
Button p, q, r;
p = new Button();
q = p; // q referencia al mismo Button
r = p.clone(); // r es una copia de p
p.setLabel("Ok"); // alterar p altera a q pero no a r
String s = q.getLabel(); // en s queda "Ok"

int i = 3;
int j = i; // j contiene una copia de i
j = 2; // ahora i == 2 y j == 3
```

5.2 Semántica de la igualdad

El operador == testea si dos variables referencian al mismo objeto, no si ambos objetos tienen el mismo valor. Para testear si dos objetos tienen el mismo valor, el usuario debe definir su propio método. La mayoría de las clases definen un método equals() a tales efectos. Es responsabilidad de las clases nuevas definidas por el usuario, la implementación de algún método para testear igualdad entre objetos de la clase, cuando corresponda.

5.3 El valor nulo

Por defecto, todas las variables en PJama que no sean de tipos primitivos tienen el valor null. El valor null es un valor reservado que indica una ausencia de referencia [Fla96], es decir una variable que no referencia a objeto o array alguno. Como observación importante, null es una excepción a las reglas de tipado fuerte del lenguaje pues puede asignarse a cualquier objeto o array. El valor null no puede asignarse a variables de tipos primitivos.

6 Relaciones entre Tipos

6.1 Equivalencia

En PJama la noción de *tipo* y *clase* es la misma. El lenguaje utiliza declaraciones de clases y subclases. La noción de "tipo" simplemente se asocia a los tipos primitivos. No se maneja el concepto de *equivalencia de clases*: dos clases diferentes, aun definiendo estructuras isomórficas para sus instancias, no se consideran equivalentes entre sí.

6.2 Mecanismos de modularización

El lenguaje dispone de mecanismos de modularización basados en el concepto de *package:* Un grupo de clases relacionadas entre sí a través de la visibilidad que existe entre los elementos de ellas. En PJama, cada clase compilada se almacena en un archivo .class separado. Un programa fuente PJama tiene la extensión .java y consiste en la definición de una o varias clases(una de las cuales debe declararse public, a los efectos de poder ser accedida desde afuera del package). La sentencia package debe ser la primera en el programa fuente.

El concepto de encapsulamiento está disponible en PJama. El acceso (visibilidad) a los packages, clases y atributos están claramente definidos y pueden gestionarse mediante el uso de declaraciones especiales (public, private, protected). El siguiente cuadro [Fla96] muestra cuál es la visibilidad de los campos definidos en las clases, en diferentes situaciones:

CAMPO	public	default	protected	private protected	private
Accesible a una no-subclase desde el mismo package?	SI	SI	SI	NO	NO
Accesible a una subclase del mismo package?	SI	SI	SI	NO	NO
Accesible a una no-subclase de diferente package?	SI	NO	NO	NO	NO
Accesible a una subclase de diferente package?	SI	NO	NO	NO	NO
Heredado por una subclase del mismo package?	SI	SI	SI	SI	NO
Heredado por una subclase de diferente package?	SI	NO	SI	SI	NO

7 Clases y subclases

La relación *subclase* se define explícitamente mediante la declaración extends. Todas las clases que se definan deben tener una única superclase; en caso de no especificarse una superclase, se asume la clase Object (que es la única clase que no tiene superclase y define métodos que pueden invocarse por cualquier objeto). De esta forma, se construye una jerarquía de clases en forma de árbol, con la clase Object en la raíz. PJama solamente soporta herencia simple. Se puede especificar que una clase no pueda especializarse en una nueva subclase usando la declaración final.

Cuando una clase hereda de otra, los posibles conflictos de nombres se resuelven utilizando el nombre más especializado. Adicionalmente, es posible referenciar a métodos de la superclase usando super. Elementos de la instancia corriente se mencionan usando this.

El agregado de objetos a una clase se realiza mediante el método new() que es definido automáticamente para cada clase. Sin embargo, no existe método para borrar un objeto: el borrado se realiza en forma automáticamente mediante el mecanismo de "garbage collection" usual de Java.

8 Aspectos de bases de datos

8.1 Persistencia

En PJama, la noción de clase no tiene ninguna relación con la persistencia. Es decir, a diferencia de otras propuestas que claramente diferencian la noción de tipo de la de clase -donde los tipos definen aspectos intensionales y las clases definen aspectos extensionales- en PJama las clases y tipos son el mismo concepto, y la persistencia puede aplicarse a cualquier objeto y debe hacerse en forma explícita a través de métodos disponibles en la clase PJavaStore. Esta clase de PJama provee los métodos para lograr la persistencia de los objetos, que deben almacenarse en un objeto de clase PJavaStore. Cuando un objeto pasa a ser persistente, a dicho proceso se lo denomina *promoción* [ADJ*96]. Cuando un objeto es persistente por sí mismo, se lo denomina *raíz de la persistencia*; en otros casos un objeto es promovido automáticamente cuando un objeto que lo referencia es a su vez promovido. El sistema aplica la noción de *alcance* (reachability) cuando se trata de hacer persistente a un objeto: todos los objetos referenciados automáticamente pasan a ser persistentes.

Todas las operaciones de manipulación de los objetos persistentes se realiza mediante los métodos expuestos en la clase PJavaStore. El borrado de objetos persistentes puede hacerse mediante métodos específicos o también mediante algoritmos automáticos de "disk garbage colection". Adicionalmente, durante el borrado de objetos persistentes se aplica el alcance: un objeto persistente no es borrado si hay algún otro objeto persistente referenciándolo.

8.2 Transacciones

En los prototipos desarrollados solamente se dispone de mecanismos rústicos para realizar un control transaccional. Por ejemplo, se dispone de un método stabilizeAll que permite aplicar un checkpoint global a toda la aplicación [ADJ*96]. Hay indicios de que se está considerando el disponer de transacciones largas y transacciones anidadas, mediante el uso de dos APIs (Application Programming Interface), una que otorga una vista funcional del manejo transaccional (un modelo transaccional sencillo) y otra que permitirá al programador extender a su gusto el modelo transaccional disponible.La primera se denomina *API transaccional interna* y la segunda se denomina *externa*. Esta última puede resultar de una instancia de la API interna. En la API interna se dispone de varias primitivas transaccionales como UpdateBookKeeper (permite implementar un *log* de transacciones, realizar *snapshots* y operaciones *undo* y *redo*) y LockingCapability (que representa a los *locks* acumulados) [AJD*96].

El manejo transaccional por defecto se apoya pues únicamente en estabilización global, que se ejecuta mediante una invocación implícita automática al método stabilizeAll(), al finalizar la ejecución del programa. Si el programa falla en manejar una excepción, terminará su ejecución dejando completamente incambiado al "store". Esto es en cierta medida equivalente a un ABORT. En los demás casos, el programa PJama actúa como una única transacción que comienza cuando se ejecuta el primer método y termina dando COMMIT cuando termina normalmente la ejecución.

8.3 Mecanismos de Recuperación

El programador puede especificar su propio mecanismo de recuperación, en el método main() de alguna clase. La clase PJavaStore dispone de algunos métodos, como setGlobalCallBacks y setClassCallBacks, que permite establecer una secuencia de call-backs para implementar una rutina de recuperación frente a una falla. A tales métodos se les pueden pasar parámetros para ejecutar un reintento, un *restart* normal o un *restore*.

8.4 Metadatos

La posibilidad de disponer de un depósito centralizado con información similar a los catálogos de los DBMS relacionales no está disponible, pero parece haber tendencia en los desarrolladores de PJama a disponer de metadatos almacenados en el mismo "store" que los datos que describen. La no disponibilidad de un repositorio centralizado de metainformación puede dificultar notablemente varios aspectos, incluyendo la gestión de la evolución del esquema de la base de datos.

8.5 Otros aspectos

No parecen haberse considerado aún cuestiones de seguridad como ser cuentas de usuarios, grupos, privilegios o similares. Tampoco se dispone de un lenguaje interactivo de acceso a los datos, que aparentemente debería hacerse únicamente a través de la codificación de programas PJama.

9 Resumen y Conclusiones

PJama es una implementación de objetos persistentes de Java. La implementación de la persistencia consiste

en una clase que se utiliza desde Java, sin que el lenguaje sufra cambios notorios desde la óptica del programador. La clase, denominada PJavaStore dispone de métodos para hacer persistente a cualquier objeto (se dice que el objeto "es promovido") y mecanismos para hacer que un objeto deje de ser persistente. Teniendo en cuenta que un lenguaje con tales características resulta de interés para la comunidad de bases de datos, y específicamente tiene vínculos con el tema de las bases de datos orientadas a objetos, se ha realizado un estudio de las características presentes y proyectadas de PJama. El contexto de dicho estudio se basó en las cuestiones consideradas en el artículo [ADG*89].

El sistema de tipos de PJama depende fundamentalmente del sistema de tipos de Java. Bajo esa circunstancia se ha realizado el presente estudio de las características de PJama. Los aspectos del sistema de tipos se estudiaron para el lenguaje Java. Los aspectos concernientes a la persistencia de objetos se estudiaron desde la óptica de las bases de datos, en el supuesto de que un lenguaje como Java dotado de persistencia puede ser un buen punto de arranque para un sistema de base de datos orientado a objetos. Sin embargo, parece ser que algunas características deseables para un sistema de bases de datos aún no han sido claramente resueltas en PJama. Notablemente, el modelo transaccional ofrecido y aspectos relacionados con metadatos de la base ofrecen funcionalidades demasiado simples, comparadas con las habituales en los sistemas de bases de datos. De todas maneras, PJama parece disponer de un aceptable mecanismo de almacenamiento persistente y una API que puede extenderse a los efectos de servir como arquitectura básica para la construcción de un sistema apto para gestionar una base de objetos.

Referencias

- [ADG*89] Albano, A., A. Dearle, G. Ghelli, C. Marlin, R. Morrison, R. Orzini, D. Stemple**. A Framework for Comparing Type Systems for Database Programming Languages**. *Second Intl. Workshop on Database Programming Languages.* Oregon, June 1989.
- [ADJ*96] Atkinson, M. P., L. Daynes, M.J. Jordan, T. Printezis, S. Spence. **An Orthogonally Persistent Java.** *ACM Sigmod Record,* Volume 25, Number 4, December 1996.
- [AJD*96] Atkinson, M.P., M.J. Jordan, L. Daynes, S. Spence. **Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system.** *Seventh International Workshop on Persistent Object Systems (POS7).* Cape May, New Jersey, May 1996.
- [CW85] Cardelli, L., P. Wegner. **On Understanding Types, Data Abstraction, and Polymorphism.** *ACM Computing Surveys,* Vol. 17, No. 4, December 1985.
- [Fla96] Flanagan, D. **Java in a Nutshell.** O'Reilly, 1996.
- [Jor96] Jordan, M.J. **Early Experiences with Persistent Java.** *The First International Workshop on Persistence and Java(tm) (PJW1).* Drymen, Scotland, September 1996.
- [Mor96] Morrison, M. **Java Language Fundamentals.** En *Java Unleashed.* Sams Net, 1996.
- [PAD*97] Printezis, T. , M. Atkinson, L. Daynes, S. Spence, P. Bailey. **The Design of a New Persistent Object Store for PJama.** *The Second International Workshop on Persistence and Java(tm) (PJW2).* Half Moon Bay, California, August 1997.

Internet

- Sun Microsystems Laboratories [http://www.sunlabs.com/]
- Sun Microsystems Laboratories, PJama Forest Project [http://www.sunlabs.com/research/forest/]
- PJava Project at Dept. of Computing Science, University of Glasgow [http://www.dcs.gla.ac.uk/pjava/]