25

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"A C A T L Á N"

EL FUTURO HOY EN LA PALMA DE TUS MANOS CON JAVA Y TARJETAS INTELIGENTES

T E S I N A

que para obtener el Título de

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

presenta

SERGIO PÉREZ COLÍN

asesorado por

Ing. Silvio Lanaza Hernández

México D. F. Septiembre del 2000

El Futuro hoy en la palma de tus manos con Java y Tarjetas Inteligentes

ÍNDICE

Capitulo II Tarjetas Inteligentes

AGRADECIMIENTOS

Dedico este trabajo con mucho cariño y agradecimiento:

A mis Padres

Que con sus consejos lograron sembrar en mi la semilla de la persistencia, responsabilidad, respeto y amor hacia todos los que me rodean y a mí mismo. A su apoyo incondicional y a su confianza inquebrantable de creer que podía llegar a ser lo que soy ahora y mucho más. A su amor irreprochable que realmente fue el motor que me impulsó a iniciar mi etapa educacional y hasta culminar este trabajo.

A mis Hermanos

Por regalarme su apoyo, sus regaños y consejos en los momentos más difíciles de mi vida y que gracias a ellos supe lo que era no darse por vencido ni detenerse por nada.

A mis Tíos y Primos

Por creer en mí y por motivarme a seguir adelante y sobre todo por brindarme su corazón.

A mi Futura Esposa

Por quererme y ayudarme en los momentos difíciles, por permitirme darle algún consejo o un abrazo cuando más lo necesitó, por abrirme su corazón y permitirme que entrara en él, pero sobre todo por estar siempre conmigo peleando hombro con hombro en todos nuestros problemas y diversiones. Y que pensando en vivir un hermoso futuro a su lado me motivo a concluir este trabajo.

A mis Amigos

Por darme esos momentos felices, por su ayuda incondicional durante todo este tiempo que gracias a dios hemos pasado juntos.

A la UNAM y mis Maestros

Por darme la formación y los medios necesarios para terminar mi carrera profesional.

Pero sobre todo a DIOS

Por darme la fortuna de tener a quien agradecerle todo lo que soy y por darme la dicha de tener a esta familia y amigos. Gracias Dios por darme la vida y por estar siempre conmigo.

INTRODUCCIÓN

World Wide Web, a lo largo de su existencia, ha sido un medio para distribuir información pasiva a un amplio número de personas. El Web (como se le conoce en el ámbito computacional), de hecho, ha sido bueno para ese propósito de manera excepcional. Con la adición de formularios y mapas de imágenes, las páginas Web comienzan a volverse interactivas, aunque con frecuencia, la interacción era solo una nueva manera de llegar a la misma información. Las limitaciones de la distribución en el Web eran aparentes una vez que los diseñadores empezaron a intentar extender la frontera de lo que el Web podía hacer. Incluso otras innovaciones, como el impulso del Browser (Visualizador para ver las páginas del Web) para crear animaciones dinámicas, eran sólo trucos inteligentes sobre una estructura que no estaba construida para aceptar algo más que documentos estáticos con imágenes y texto.

Con el crecimiento del Web aparecen los hackers, también denominados piratas de la Red. Estas personas se dedican a defraudar los sistemas activos en la red, pudiendo llevar a cabo pérdidas monetarias a las compañías dueñas de estos sistemas. Se ha tratado de solucionar este tipo de problemas diseñando, con nuevas tecnologías, nuevas formas de luchar contra ellos.

Una de las formas creadas y utilizadas es la tarjeta inteligente. Las tarjetas inteligentes se pueden dividir, a grandes rasgos, en dos: en tarjetas de chip y tarjetas de banda magnética. Las tarjetas de banda magnética son ya ampliamente conocidas (tarjetas de crédito o de débito). Las menos conocidas son las de chip, aunque son las más seguras (Tarjetas Telefónicas).

Dejando información confidencial dentro de las tarjetas se puede asegurar que la información contenida está protegida contra los hackers, ya que sólo puede ser accesada por medio de una llave y un password inviolable.

Dentro de las tarjetas se pueden almacenar desde datos de tipo binario, como imágenes o simplemente claves, hasta información en código ASCII, donde la capacidad de almacenaje depende puramente de la capacidad de la tarjeta.

Otra forma creada para luchar contra los hackers es la de encriptar la información que viaja sobre la red. Tomando en cuenta esta característica, tenemos entonces un lenguaje que es capaz de encriptar por sí solo, la información. Aquí aparece Java y la capacidad de las páginas Web de contener los applets Java.

Los applets son pequeños programas que crean animaciones, presentaciones con multimedia, juegos en tiempo real (video), juegos para múltiples usuarios en red e interactividad real; de hecho todo lo que puede realizar un pequeño programa, los applets de Java lo pueden hacer. Bajados de la red y ejecutados dentro de una página Web por un visualizador que acepta Java, los applets son un gran paso adelante en el diseño del Web estándar.

Con la alianza de estas dos tecnologías se puede tener o diseñar un sistema con un alto grado de seguridad y desempeño, ya que si se tienen en cuenta las características mencionadas anteriormente se puede lograr un sistema donde una parte de la información viaje encriptada y otra pueda estar siempre con el usuario, es decir, en una tarjeta inteligente. De modo que para poder afectar a este sistema se tendría que tener, por un lado, el uso de la aplicación y, por el otro, la tarjeta puesta en lector para accesar los datos, ya que sólo así se podría hacer uso del sistema.

En este trabajo se presenta un ejemplo real de una aplicación donde se utilizan ambas tecnologías (Java y tarjetas Inteligentes), haciendo notar cómo la seguridad, tanto en software como en hardware, es muy alta. La aplicación transporta de forma protegida y confiable los datos de una persona que está asegurada, para lo cual comparte información con médicos, farmacéuticos y con personal directo de la aseguradora. La idea general de esta aplicación es la de tener

un mejor control de las personas aseguradas por la compañía y de evitar los actos ilícitos a ésta.

Cabe mencionar además que en este trabajo la finalidad no es la de analizar a fondo el sistema que se plantea en el Capítulo III, "Autentificación de usuarios mediante Tarjetas Inteligentes", si no más bien la de hacer notar cómo, en conjunto, Java y las tarjetas inteligentes, pueden ser de gran ayuda para obtener seguridad y sencillez en las aplicaciones que se realicen, así como para incrementar la velocidad en tiempo desarrollo. Es importante también recalcar que este material esta enfocado a personas con cierta experiencia en el ramo de la computación y en el desarrollo de sistemas de información basado, preferentemente, en el paradigma de Orientado a Objetos, ya que el lenguaje de programación que se menciona en este trabajo de Tesina, utiliza dicho paradigma al 100% en un ambiente de Web.

CAPÍTULO I

EL LENGUAJE JAVA

1.1 ¿Qué es Java?

Java es un lenguaje:

- Simple
- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Portable
- Interpretado
- Alto - Desempeño
- Multihilos (Multithreads)
- Dinámico

Simple

El lenguaje Java ofrece toda la funcionalidad de un poderoso lenguaje. Fue diseñado para ser como C++, para un rápido y fácil aprendizaje.

El lenguaje Java elimina muchas de las características de otros lenguajes de objetos, como C++, para permitir un código pequeño y además una característica muy especial como lo es la recolección automática de basura. El lenguaje es tan pequeño que su intérprete sólo utiliza alrededor de 215 Kb de RAM.

Orientado a Objetos

El paradigma orientado a objetos se basa en los tres métodos de organización que el hombre utiliza desde su infancia y en los que se basa todo su pensamiento: la diferencia entre un objeto y sus atributos (por ejemplo, entre un camión y su altura o peso), la diferencia entre un objeto y sus componentes (por ejemplo, entre un camión y sus ruedas o motor) y la formación y distinción entre clases de objetos (por ejemplo, entre camiones, turismos y bicicletas).

Un objeto es la abstracción de algo que forma parte del dominio del problema, reflejando las posibilidades de un sistema para mantener información sobre él. Representa una entidad, real o abstracta, con un papel bien definido dentro del mundo y con dos características: su estado y su comportamiento. Por ejemplo, una bicicleta.

Los objetos se describen mediante atributos o variables, que aclaran el significado del objeto. Un atributo de un objeto es algo que el objeto conoce y refleja el estado del objeto. Cuando un objeto necesite conocer atributos de otros objetos del modelo estos podrán asociarse mediante un tipo de conexión denominada asociación (o interconexión). Tomando el ejemplo anterior, los atributos del objeto bicicleta podrían ser: el número de llantas que tiene, la velocidad en cada momento, el número de pedales, etc.

Para alterar su estado los objetos necesitan métodos. Un método de un objeto es algo que el objeto sabe hacer y define el comportamiento general del objeto. Los métodos modifican uno o más atributos de los objetos. El concepto de método de la programación orientada a objetos es similar al concepto de procedimiento o función de la programación imperativa. Tomando el objeto bicicleta como ejemplo, alguno de sus métodos podrían ser: acelerar, frenar, girar, etc.

La idea de manejar objetos reales como contenedores de estados y comportamientos es mucho más atractiva desde el punto de vista del usuario. Así no tendrá que batallar con construcciones orientadas a la computadora, sino que podrá manejar objetos (y operaciones) que se asemejen más a sus equivalentes en el mundo real. La elevación del nivel de abstracción es sin duda un objetivo deseable.

Las técnicas orientadas a objetos usan un mismo modelo conceptual para el análisis, el diseño y la programación. La transición desde el análisis al diseño es tan natural que es difícil especificar donde comienza uno y donde acaba el otro.

Las ventajas en la utilización de este paradigma son claras:

- Favorecerá la comunicación entre analistas, diseñadores, programadores y usuarios finales al utilizar todos el mismo modelo conceptual.
- El favorecer la comunicación se traducirá en un aumento de la productividad, ya que la comunicación es uno de los puntos críticos en las primeras fases del proyecto.
- Es posible representar una estructura compleja sin necesidad de normalizar, ya que lo que manejamos son objetos del mundo real, lo que facilitará la tarea del analista.
- La semántica de estas técnicas es más rica (al ser más natural); al usuario final le será más fácil comprender lo que el analista representa en sus modelos (ya que representa los objetos que le rodean habitualmente).
- Favorecerá la modularidad, la reusabilidad y mantenibilidad del software.
- Estas técnicas son más resistentes al cambio que las tradicionales técnicas de análisis orientadas a flujos de datos.

El lenguaje Java implementa la tecnología básica de objetos de C++, con algunas modificaciones. Algunas de las implementaciones son: 1) en el lenguaje Java no se utilizan apuntadores, a nivel del programador, sólo internamente; 2) en el lenguaje Java se utilizan todos los conceptos de Orientado a Objetos, como son herencia, polimorfismo y encapsulamiento, entre otros, 3) no utiliza la herencia múltiple, sólo herencia simple.

Distribuido

El lenguaje Java tiene un constructor extensivo TCP/IP para las redes. Existen rutinas de librerías para accesar e interactuar con protocolos tales como HyperText Transfer Protocol (HTTP) y File Transfer Protocol (FTP) con el objeto de recuperar información. Esto permite a programadores accesar información a través de la red con la misma facilidad que la de accesar archivos locales.

Robusto

El lenguaje Java es poderoso para verificar problemas en la compilación y tiempo de corrida. El tipo de chequeo en Java ayuda a atrapar errores tempranamente en el ciclo de desarrollo. Maneja las fallas de memoria. Además, Java implementa verdaderos arreglos para evitar la posibilidad de datos dañados. Estas características disminuyen drásticamente el tiempo de desarrollo de aplicaciones Java.

El lenguaje Java provee:

- Verificador de punteros nulos.- Verifica que no haya referencias a direcciones de memoria nulas.
- Verificador de arreglos.- Verifica que cada uno de los elementos del arreglo no apunten hacia una dirección nula.
- Excepciones.- Definen condiciones de error medio que los programas puedan encontrar.
- Verificación de Bytecodes.- Verifica que el código que va a ser interpretado tenga el formato correcto.
- Recolección automática de basura.- Libera automáticamente las direcciones en memoria que no estén en uso.

Seguridad

El lenguaje Java incrementa la seguridad por la eliminación de punteros y por el cambio de propiedades que se les puedan hacer a los objetos y a otros tipos de dato en tiempo de corrida dentro de C y C++, para prevenir acceso ilegal a memoria. En adición, la seguridad es provista por el intérprete de Java dentro de la verificación de código de la Java Virtual Machine (JVM). El funcionamiento de la JVM se explica ampliamente en el tema 1.2.1 Arquitectura Neutral.

Un intérprete de Java debe ejecutar código compilado por la JVM. Un intérprete Java tiene tres tareas principales:

1. Cargar código.- Hecho por el cargador de clases.
2. Verificación de código.- Hecho por el verificador de bytecodes.
3. Ejecución de código.- Hecho por el sistema al tiempo de corrida.

El cargador de clases carga todas las clases necesarias para la ejecución de un programa.

El verificador de bytecode examina detalladamente el código Java ejecutado en la máquina. La máquina corre el programa a través del verificador de bytecodes que examina el formato de fragmentos de código para código ilegal.

Portable

Más allá de los principios de la portabilidad, el lenguaje Java implementa otras portabilidades estándares para facilitar el desarrollo de aplicaciones. El lenguaje Java construye interfaces de usuarios a través de un sistema de ventanas abstractas, de tal forma, que puedan ser implementadas sobre ambientes UNIX, PC y MAC.

Interpretado

El intérprete de Java puede ejecutar directamente instrucciones en el código objeto.

Alto – Desempeño

El compilador Java optimiza automáticamente el código objeto. Sun Microsystems reporta 300,000 llamadas de métodos o procedimientos por segundo usando el intérprete correcto (sobre una SPARCStation 10).

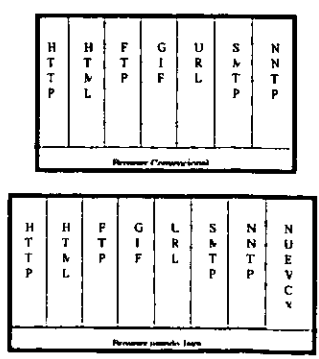
Multihilos (Multithreads)

Los multihilos Java permiten muchas tareas simultáneas. Los hilos son básicamente pequeños procesos o piezas independiente de grandes procesos. Ya que los hilos son construidos dentro del lenguaje Java, son más fáciles de usar y más robustos que sus contrapartes en C o C++.

Dinámico

El lenguaje Java toma ventaja de muchas tecnologías de objetos tanto como es posible. El lenguaje Java no trata de conectar todos los módulos que componen una aplicación hasta el tiempo de corrida. Las librerías nuevas o actualizadas no chocarán con las aplicaciones actuales, sino que proveerán mantenimiento a las API's básicas. Es decir, actualmente las aplicaciones hacen uso de las librerías estándares en los browsers para utilizar determinados tipos de archivos, pero cuando se utiliza Java se pueden crear nuevas librerías sin que se afecten las ya existentes, para utilizar otros tipos (figura 1.1).

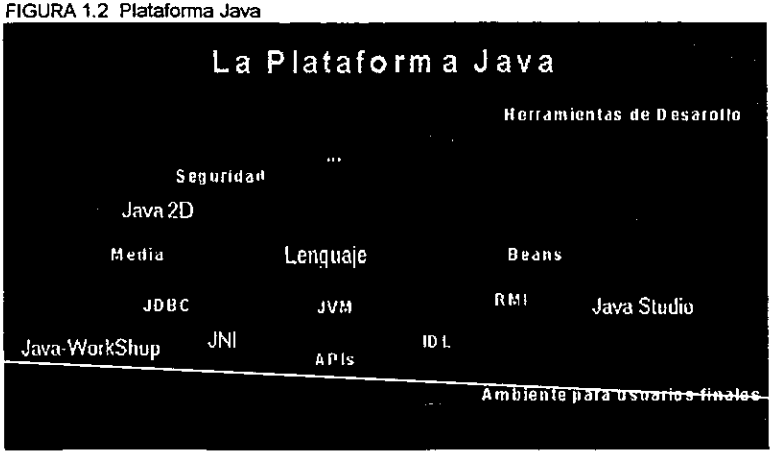
FIGURA 1.1 Tipo de archivos en un Browser



1.2 Arquitectura

Como se ha visto, el lenguaje Java tiene una arquitectura 100% basada en objetos (figura 1.2):

FIGURA 1.2 Plataforma Java



Su núcleo se compone de lo que es el lenguaje de programación en sí; es aquí donde está construido básicamente el JDK (Java Developer Kit) que es la herramienta primaria de desarrollo. Esta herramienta se puede "bajar" de Internet del siguiente sitio: http://www.java.sun.com/products
Ahí se encontrará la última versión del lenguaje además de versiones posteriores. El software es gratuito.

La segunda elipse trata de la JVM (Java Virtual Machine), que se encarga de hacer la interpretación del código en el Browser, como se verá en el siguiente tema.

Más externamente se tienen, las características del lenguaje o herramientas de uso interno. Por ejemplo, se tiene: JDBC (conectar a Bases de Datos mediante Java), RMI (Invocación de métodos remotos), Beans (son objetos construidos en Java para desarrollar aplicaciones Java), entre otros.

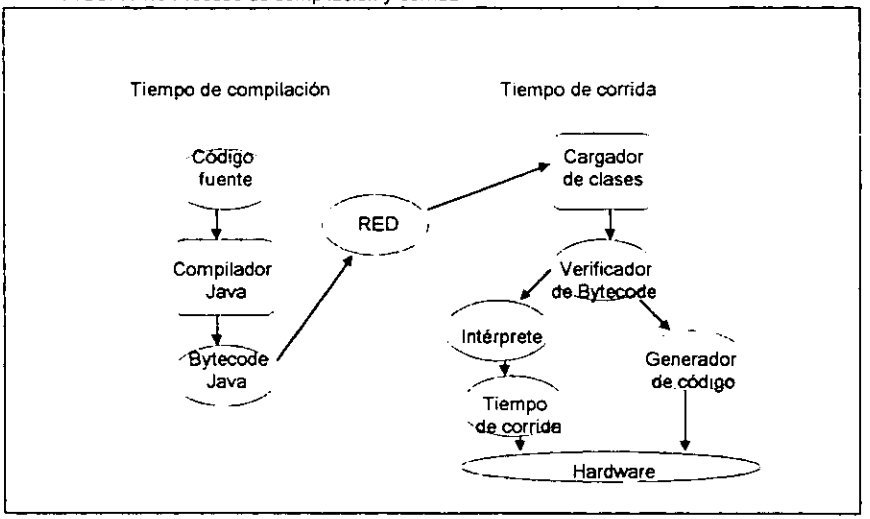
En la parte más externa de las elipses están lo que son los productos visuales para desarrollar en Java, como: Java-WorkShop, Java Studio, etc.

Y por último, lo que es el ambiente para usuarios finales, es decir la creación de aplicaciones a la medida hechas por desarrolladores de sistemas.

1.2.1 Arquitectura Neutral

Para establecer al lenguaje Java como una parte integral de la red, el compilador de Java, compila los códigos fuente a una arquitectura - formato de archivos de objetos independientes (figura 1.3). Cualquier máquina que tenga el sistema de tiempo de corrida de Java, puede ejecutar este código de objetos. Actualmente, hay ambiente de tiempo de corrida para sistemas Solaris 2.4(SPARC), Mac y Win32 (para windows 95 y windows NT).

FIGURA 1.3 Proceso de compilación y corrida



Existen dos tiempos en Java: el tiempo de compilación y el tiempo de corrida o ejecución. En el tiempo de ejecución el código fuente es compilado en alto nivel, máquina independiente, formato Bytecode. Ya en el tiempo de ejecución o corrida, el código compilado pasa por un proceso de cargado en memoria, seguido de una verificación de formato para que posteriormente se designe su ejecución en una máquina hipotética ("Java Virtual Machine") y sea implementado por una máquina dependiente del tiempo de corrida.

La Java Virtual Machine (JVM) se define como:

Una máquina imaginaria que es implementada por emulación del software sobre una máquina real. El código para la JVM es almacenado en un archivo *.class*, cada uno de los cuales contiene código para más de una clase *public*.

La JVM provee la plataforma de hardware para la cual todo Java es compilado. Estas especificaciones permiten a los programadores estar en plataformas independientes, porque la compilación es hecha por una máquina imaginaria.

Las especificaciones de la JVM proveen definiciones concretas para la implementación de lo siguiente:

- Conjunto de instrucciones (equivalentes al conjunto de instrucciones de una CPU)
- Conjunto de registros.
- Formato de archivos .class.
- Pilas.
- Colector de basura.
- Área de memoria.

El formato del código para la JVM es compacto y con bytecodes eficientes. Programas representados por los bytcodes de la JVM deben de mantener propiamente un tipo de disciplina. La mayoría del tipo de chequeo de Java es hecho en el tiempo de compilación por los verificadores de bytecodes. Cualquier intérprete Java debe estar disponible para correr cualquier programa con archivos *.class* que conforman el formato especificado de archivos class de la JVM.

1.3 Programación Básica

Conocimientos básicos para programar en Java

Para poder escribir programas Java, es obvio que necesitará un ambiente de desarrollo Java. Se puede obtener el Java Development Kit (JDK) en diferentes lugares:

◊ El CD-ROM.
◊ La última versión del JDK puede cargarse del sitio FTP de Java de Sun en ftp://java.sun.com/pub/ o desde un sitio espejo (ftp://www.blackdown.org/pub/java/pub es uno de ellos).

Los programas Java comprenden dos grupos principales: applets y aplicaciones.

Los applets, como se sabe, son programas Java que se cargan del World Wide Web y se ejecutan mediante un visualizador Web. Para ejecutarse, dependen de un visualizador habilitado para Java. Dentro de los visualizadores más comunes están: Netscape x, MSIExplorer x y por supuesto el propio visualizador de Java, HotJava. Las aplicaciones Java son programas más generales, escritos en este mismo lenguaje. No requieren un visualizador para ejecutarse y, de hecho, Java puede emplearse para crear toda clase de aplicaciones posibles, como si utilizara un lenguaje de programación más convencional. El mismo HotJava es una aplicación Java.

A continuación se verá la creación de una aplicación Java sencilla: el ejemplo clásico Hola Mundo que todos los libros de lenguaje utilizan para comenzar.

Todos los programas que se crean en Java, se crean en un editor de texto sencillo, o en uno que puede guardar archivos en ASCII común, sin caracteres para dar formato. En UNIX, se puede trabajar con vi, emacs o pico; mientras que en Windows, Notepad o DOS Edit, son editores de texto aceptados.

Ejemplo.-

```
class holamundo {
  public static void main (string arg[]) {
    System.out.println(" Hola mundo" );
  }
}
```

Este programa tiene dos partes principales:

1. Todo el programa está encerrado en una definición de clase (aquí una clase llamada holamundo)
2. El cuerpo del programa está contenido en una rutina llamada main(). En las aplicaciones Java, al igual que en un programa en C o C++, esta es la primera rutina que corre cuando el programa se ejecuta.

Cuando se termina de teclear el programa se guarda con extensión *programa*.java. Por convención todos los archivos Java tienen el mismo nombre que la clase que definen. Por lo tanto este programa debe llamarse holamundo.java.

Ahora se compila el archivo fuente con el compilador Java. En el JDK de Sun, el compilador de Java se llama javac.

Para compilar el archivo, se debe asegurar que el programa javac se encuentra en la misma ruta de su programa (o en la variable de ambiente Path) y después teclear javac seguido del nombre de su archivo fuente:

javac holamundo.java

Cuando el programa se compila sin errores, se termina con un archivo llamado holamundo.class, en el mismo directorio de su archivo fuente. Este es el archivo bytecode de Java y se podrá ejecutar con el intérprete Java; en el JDK se llama simplemente *java*. Se debe de asegurar que el programa java se encuentre en la misma ruta que el archivo fuente y después teclear:

java holamundo

Si el programa se escribió y compiló de manera correcta, se debe de obtener la cadena: " Hola mundo", impresa en la pantalla.

Crear applets (mini versión de una aplicación que necesita de un visualizador para ejecutarse) es diferente a crear una aplicación sencilla, ya que los applets Java se ejecutan y se despliegan dentro de una página de Web con otros elementos de página y, como tales, con reglas especiales para el modo en que deberían comportarse.

Por ejemplo, para crear un applet holamundo, en lugar de sólo estar habilitado para imprimir un mensaje, tendremos que crear otro para hacer espacio para el mensaje y luego, utilizar operaciones gráficas para pintar el mensaje en la pantalla.

Ahora se teclea el siguiente código, para poder ver un applet:

```
1: import java.awt.Graphics;
2:
3: public class holamundoapplet extends java.applet.Applet {
4:     public void paint (Graphics g) {
5:         g.drawString(" hola mundo",5,25);
6:     }
7: }
```

Se guarda y se compila este archivo de la misma manera que la aplicación.

Hay un par de características sobre los applets que cabe señalar:

- La línea import al inicio del archivo es, de alguna manera, análoga al enunciado #include en C; esto le permitirá al applet tener acceso a las clases JDK para crear applets y dibujar gráficos en pantalla.
- El método paint() despliega el contenido de un applet en la pantalla. Aquí la cadena hola mundo se dibuja. Los applets utilizan varios métodos estándares para tomar el lugar de main(), los cuales incluyen init(), para inicializar el applet, start(), para empezar la ejecución, y paint(), para desplegarla en pantalla.

Para incluir un applet en una página Web, nos debemos de referir a él en el código HTML para dicha página. A continuación se crea un archivo HTML sencillo:

```
1: <html>
2: <head>
3: <title> hola mundo </title>
4: </head><body>
5: <br>Mi applet Java dice:
6: <applet code="holamundoapplet.class" width=200 height=200> </applet>
7: <body>
8: </html>
```

Se guarda el archivo con extensión *html* para poder ver el applet. Para ello se necesita escoger una de las siguientes opciones:

- Un visualizador que acepte applets Java.
- La aplicación appletviewer, la cual es parte de JDK. Puesto que no es un visualizador Web no le permitirá ver la página por completo, pero es aceptable para experimentar con la apariencia de un applet y su comportamiento, si no cuenta con otro visualizador disponible.

1.3.1 Inicio de Programación

El lenguaje de programación Java es muy similar a la manera en la que se programaría en C y C++, ya que la declaración de métodos y funciones son similares.

Para poder poner comentarios en el programa se puede hacer de la siguiente manera:

```
// comentario en una línea
/* comentario en una o más líneas */
/** para comentar todo el documento */
```

Este último genera un archivo HTML al compilar el programa con la documentación en línea de Java, javadoc.

La manera de declarar variables es muy sencilla y se terminan con punto y coma (;)

```
int x;
x =2 *9 ;
```

En Java los identificadores se pueden empezar con una letra guión bajo (_), o signo de dólar ($). Los identificadores son sensitivos, esto es que respeta si se escriben con mayúsculas o minúsculas, y no tienen longitud máxima.
Las palabras clave para Java son:

abstract	default	if	package	this
boolean	do	implements	private	throw
break	double	import	protected	throws
byte	else	instanceof	public	transient
case	extends	int	return	true
catch	false	interface	short	try
char	final	long	static	void
class	finally	native	super	while
const	float	new	switch	continue
for	null	synchronized		

Las palabras true, false y null van con minúsculas y no con mayúsculas como el C ++.

Los tipos de datos primitivos que existen en Java son:

Nombre	Longitud	Valor inicial
byte	8 bits	0
short	16 bits	0
int	32 bits	0
long	64 bits	0L
float	32 bits	0.0f
double	64 bits	0.0d
char	16 bits	'\u0000' (NULL)
boolean	8 bits	false
todos los tipos de referencias		null

La declaración de los arreglos puede ser de cualquier tipo de dato:

```
char s[ ];
int [ ] arreglo;
```

Note que la posición de los corchetes determina como se aplicará la lista;

```
int [ ] x, y[ ];
```

es igual a:

```
int  x[ ], y [ ] [ ];
```

Para crear los arreglos se puede utilizar la palabra new. Se pueden construir, además, arreglos de arreglos:

```
int tabla [ ] [ ] = new int [4] [5];
```

Si se desea conocer la longitud del arreglo, este tiene una variable llamada *length*. Se puede usar esta instancia de la variable para saber la longitud de un arreglo; del ejemplo anterior se tiene que:

```
tabla.length      /* 4 que es el número de renglones */
tabla [0].length  /* 5 el número de columnas */
```

También se pueden crear arreglos con valores iniciales:

```
int nombres[ ] = {" Marcos" , " Tomas" , " Juan" , " Simon" };
```

Los operadores que se manejan en Java son muy semejantes a los que se utilizan en el lenguaje C:

Operador	Significado
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
==	Igual
!=	Distinto
++, --	Incremento, decremento
!	Operador unario, cambia true por false y viceversa
&&	Y condicional
||	O condicional
&	Y a nivel de bits
|	O inclusivo a nivel de bits
^	O exclusivo (XOR) a nivel de bits
~	operador unario a nivel de bits, cambia cada uno de los bits
<<	Desplaza los bits a la izquierda, rellenando por el lado derecho con bits ceros
>>	Desplaza los bits a la derecha, rellenando con el bit más alto(signo) por el lado izquierdo
>>>	Desplaza los bits a la derecha, rellenando con bits cero por el lado izquierdo
?=	Operador condicional, da como resultado uno de dos valores a partir de una expresión booleana
+=, *=, -=	operadores de asignación

1.3.2 Sentencias if, else

La forma más elemental de flujo de control condicional es la sentencia if, que elige si se ejecutan o no las sentencias siguientes. Su sintaxis es como sigue:

```
if(expresión-booleana){
   sentencia;
}
else {
   sentencia;
}
```

1.3.3 Sentencia switch

Una sentencia switch evalúa una expresión cuyo valor se usa para encontrar una etiqueta *case* adecuada entre las listadas dentro de los bloques siguientes. Si se encuentra una etiqueta case correspondiente, se transfiere el control a la primera sentencia que le sigue, si no, se transfiere el control a una etiqueta *default*. Si no hay etiqueta *default* se salta toda la sentencia *switch*:

```
switch (expresión){
   case expresion1:
        sentencias;
        break;
   case expresion2:
        sentencias;
        break;
   default:
```

```
        sentencias;
        break;
}
```

1.3.4 Sentencia for

La sentencia for se usa para repetir un bucle a lo largo de un rango de valores desde el principio hasta el final:

```
for (exp_inicial; expresion_booleana; expr_incremento) {
   sentencia
}
```

1.3.5 Sentencias while y do-while

El bucle while tiene la siguiente sintaxis:

```
while(expresion_boolean)
  sentencia
```

La expresión booleana es evaluada, y si es verdadera, la sentencia se ejecuta una y otra vez hasta que la expresión booleana sea falsa. Esta se puede ejecutar de cero a n – veces.

Cuando se desea ejecutar el cuerpo de un bucle al menos una vez, se utiliza la instrucción do-while

```
do {
   sentencia
} while ( expresion-booleana);
```

Se puede utilizar *break* para salir de cualquier bloque, no sólo de un *switch*.

La sentencia *continue* salta al final del cuerpo de un bucle y evalúa la expresión booleana que controla dicho bucle. Puede utilizarse para saltarse elementos del rango del bucle.

Una sentencia *return* termina la ejecución de un método y devuelve al invocador.

1.4 Paquetes

Los paquetes son el resultado de cómo Java diseña y organiza a gran escala. Se utilizan para categorizar y agrupar clases.

El lenguaje Java contiene una serie de paquetes que incluyen propiedades de ventanas, utilidades estándar, flujos de I/O (E/S entrada/salida), herramientas y redes de trabajo.

Los paquetes son cargados con la instrucción *import* que se pone al principio del programa, especificando tanto el nombre del paquete como la ruta y el nombre de la clase. Clases múltiples pueden ser cargadas usando el asterisco:

```
import java.awt.*;
```

Para el caso de sólo querer cargar una clase:

```
import java.util.Date;
```

1.4.1 Paquete java.applet

Este paquete incluye clases diseñadas para utilizar en un applet. Esta tiene una clase Applet y tres interfaces; AppletContext, AppletStub, y AudioClip.

1.4.2 Paquete java.awt

Otro paquete con propiedades de ventanas es awt, que contiene clases usadas para crear componentes de interfaces gráficas del usuario. Este paquete incluye las clases: Button, CheckBox, Choice, Component, Graphics, Menu, Panel, TextArea, y TextField, entre otras.

1.4.3 Paquete java.io

El paquete I/O incluye clases de entrada y salida, tales como FileInputStream y File OutputStream. Los archivos de I/O están sujetos a control de seguridad en los applets.

1.4.4 Paquete java.lang

Este paquete incluye las clases del lenguaje de Java, como son Object, Thread, Exception, System, Integer, Float, Math, String y otras. Estos paquetes son automáticamente cargados por el compilador de Java y no necesitan ser importados para el uso de estas clases.

1.4.5 Paquete java.net

Este paquete soporta protocolos de red TCP/IP e incluye Socket, DatagramPacket, DatagramSocket, URL y URLConnection, entre otras clases.

1.4.6 Paquete java.util

Contiene diversas clases que son usadas por una variedad de tareas de programas. Estas clases incluyen Date, Dictionary, Random (para crear números aleatorios), Vector, y Stack (para implementar listas ligadas).

1.5 Principales clases y métodos

A continuación se verán las clases y métodos más usados para la elaboración de un programa. Cabe hacer notar que estos no son todos los que existen y que habrá que hacer uso de nuestra capacidad de investigación para aprender a utilizarlos.

1.5.1 Clase Applet

Un applet es una mini versión de una aplicación en Java, esto es, un programa que se ejecuta en un visualizador. El applet asume que el usuario está ejecutando código en Java. Para poder utilizar un applet la siguiente línea debería aparecer en el código:

public class *nombre_clase* extends Applet {

Para poder correr el applet en un visualizador se necesita de una página HTML donde se especifique el applet que va a ser cargado. Los requerimientos mínimos para poder cargar el applet son:

```
<HTML>
<applet code=HolaMundo.class width=100 heigth=100>
</applet>
</HTML>
```

Donde:

code=HolaMundo.class está indicando a la página HTML el nombre de la clase que necesita ejecutar para correr el applet.

width=100 está indicando a la página que el applet va a tener un ancho de 100 pixeles.

height=100 indica que va a tener un alto de 100 pixeles.

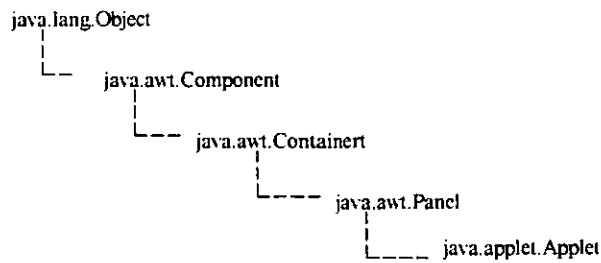
1.5.1.1 Ciclo de vida de un Applet

Esto es el orden en el que son llamados los métodos al momento de ser cargado el applet, donde el orden de llamada es el siguiente:

init()
paint(g)
start()
stop()
update(g)
destroy()

Estos métodos pueden ser modificados programándolos según las necesidades.

1.5.1.2 Jerarquía de Clases del Paquete java.applet



Para poder crear un applet es necesario importar las clases de Applet además de extender a Applet la clase que se va a crear, como se muestra en el ejemplo siguiente:

```
import.java.applet.Applet;
. . .
public class HolaMundo extends Applet {
....
}
```

Los métodos que son utilizados para empezar y detener un applet son los siguientes:

1.5.1.3 El método init()

Esta función es ejecutada al tiempo en que el applet es creado y primero cargado en el visualizador. El método init es llamado entre los métodos stop y start y después entre stop y destroy. El applet no hace nada en esta función. Se puede sobreescribir este método y redimensionar el applet, asignar valores a las variables. Sólo es un método de inicialización.

Por ejemplo:

```
public void init(){
  if ((size().width <200) || (size().height < 200)){
    resize(200,200);
  }
global_value1 = 0;
global_value2 = 100;
}
```

1.5.1.4 El método start()

Este método inicia el applet. Es ejecutado cuando un applet es visitado. Este método puede ser sobreescrito para empezar animación, sonido, entre otros.

```
public void start() {
 isStopped = false ;
 //empieza la música
 musicClip.play();
}
```

1.5.1.5 El método stop()

Este método detiene el applet. Es ejecutado cuando el applet ya no está en la pantalla.

```
public void stop(){
  isStopped = true;
  if ( /* la música está sonando? */ )
  {
    musicClip.stop();
  }
}
```

1.5.1.6 El método destroy()

Este método es ejecutado cuando el visualizador existe. Si se modifica este método se puede limpiar la pantalla al final. Los multithreads o multihilos pueden utilizar este método para detener algún thread vivo.

1.5.2 El método paint (Graphics g)

Es ejecutado cuando el applet es dibujado y este puede ser refrescado. Además de ser usado por el applet este método es utilizado por la jerarquía de clases Component. En este ejemplo el método paint dibuja una caja en 3-D en el área. Este método puede ser modificado y dibujar cualquier área gráfica.

```
public void paint (Graphics g) {
  g.drawString("Hola Mundo! ", 25,25);
  //puedes dibujar o mostrar la imagen que se requiera
}
```

1.5.3 Otros métodos para gráficas

Estos son otros métodos que se pueden utilizar para poder dibujar, únicamente se necesitan importar las clases java.awt.Graphics:

- drawLine(int x1, int y1, int x2, int y2)
- drawRect(int x, int y, int ancho, int largo)
- drawOval(int x, int y int ancho, int largo)

Para los métodos drawRect y drawOval, las coordenadas x, y están en la esquina superior izquierda.

Para poder utilizar información que haya sido cargada en la página html, de alguna variable se utiliza el método getParameter(String str) si el parámetro en la página HTML no tiene ningún valor, regresa nulo.

Ejemplo:

```
public void init(){
   String pv;
   pv=getParameter("speed");   // busca el valor del parámetro llamado "speed" en
                                  la página HTML

   if (pv ==null){
      speed=10;
   }
   else{
     speed=Integer.parseInt(pv);   // convierte un String a un entero
   }
  }
```

Para cargar imágenes, ya sea de alguna dirección, o un archivo o de la misma página HTML se utiliza el método getImage(URL , String), Ejemplo:

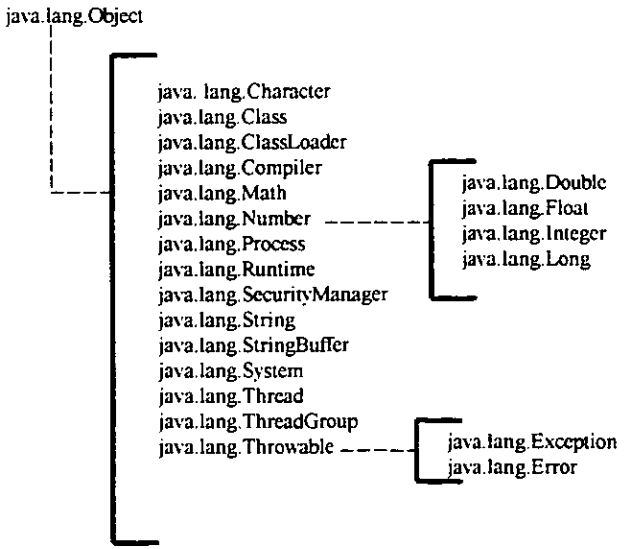
```
public void init(){
   Image im;
   im=getImage(getDocumentBase(),"duke.gif");
}
```

```
public void paint(Graphics g){
   g.drawImage(im,x,y,this);
}
```

Para cargar un audio clip se utiliza el método getAudioClip, método de la clase java.applet.Applet, que además se le puede asociar uno de estos tres métodos play, loop o stop.

1.5.4 Jerarquía de Clases del paquete java.lang

Este paquete es cargado automáticamente por el compilador de Java, por lo tanto no se necesita importar para usar las clases del paquete. Las clases de las que se compone este paquete son las siguientes:



Las clases Character, Integer, Long, Float, Double y Boolean son tipos de datos tratados como objetos, que es la manera como se definen en programación orientada a objetos, estos también cuentan con métodos para que puedan operar en su forma primitiva (como sería una declaración en C o Pascal).

A continuación se detallarán únicamente las clases que se consideran más importantes por su uso.

1.5.4.1 Clase Math

Los métodos de la clase Math son llamados usando la sintaxis *Math.método()*. Algunos de los métodos más usados son los siguientes:

- public static *tipo* abs(type x) Regresa el valor absoluto del valor de x. donde el *tipo* puede ser un int, long, float, o double.
- public static double sqrt(double x). Regresa la raíz cuadrada de x, donde $x \geq 0.0$
- public static double pow (double x, double y). Regresa x^y.
- public static synchronized double random(). Genera números aleatorios entre 0.0 y 1.0.
- public static *tipo* max(tipo a, tipo b). Regresa el mayor de los dos números.
- public static long round (double x). Redondea el valor si es mayor de .5 al entero mayor.

1.5.4.2 Clase Character

Para definir una variable de esta clase se hace de la siguiente manera:

```
char c;      /* cuando se quiere una variable de tipo caracter*/
Character C  /*  cuando se declara un objeto de tipo Character*/
```

Algunos de los métodos de la clase Character son los siguientes:

- public static boolean islowerCase(char ch). Determina si el caracter especificado está en minúscula.
- public static boolean isUpperCase(char ch). Determina si el caracter especificado está en mayúscula.
- public static boolean isDigit(char ch). Determina si el caracter especificado es un dígito.
- public static boolean isSpace(char ch). Determina si el caracter especificado es un espacio en blanco de acuerdo al lenguaje Java Puede ser un espacio en ASCII, tabulador horizontal, regreso de carro, etc.
- public static char toLowerCase(char c). Regresa el caracter especificado en minúscula.
- public static char toUpperCase(char c). Regresa el caracter especificado en mayúscula.
- public static int digit(char ch, int radix). Regresa el valor numérico del caracter usando el dígito especificado en la base. Si el caracter no es un dígito válido regresa -1.
- public static char forDigit(int digit, int radix). Lo mismo que el método anterior, pero si el dígito no es válido en la base, regresa cero.
- public char charValue(). Regresa el valor de este objeto Character.

- public String toString(). Regresa un objeto String representado por el valor del caracter.

1.5.4.3 Clase Integer

Para poder declarar una variable u objeto de esta clase la notación es la siguiente:

```
int i;     /* para declarar una variable entera */
Integer I; /* para declarar un objeto entero */
```

Se verán ahora algunas variables de la clase Integer

- public final static int MIN_VALUE . El mínimo valor entero que puede tener es (0x80000000).
- public final static int MAX_VALUE. El máximo valor entero que puede tener es (0x7fffffff).

Algunos de los métodos que se utilizan en la clase Integer son:

- public static String toString(int i). Regresa un nuevo objeto String representando al entero especificado.
- public static int parseInt(String s). Asumiendo que el String especificado representa un entero, regresa un nuevo objeto Integer. La base que se asume para la conversión es la 10.
- public int intValue(). Regresa el valor de ese objeto Integer como una variable int.
- public long longValue(). Regresa el valor de ese objeto Integer como una variable long.

- public float floatValue(). Regresa el valor de ese objeto Interger como un float.
- public double doubleValue(). Regresa el valor de ese objeto Integer como una variable double.
- public String toString(). Regresa un objeto String representando el valor del objeto Integer.
- public boolean equals(Object obj). Compara este objeto con un objeto especificado, regresando verdadero si el valor o contenido de los objetos es el mismo.

1.5.4.4 Clase Float

La clase Float extiende de la clase Number. Para declarar una variable de este tipo la notación es la siguiente:

```
float f;   /* para una variable float*/
Float F;   /* para un objeto Float */
```

Sus constructores se definen de la siguiente manera:

- public Float(tipo valor). Construye un objeto Float para el tipo de valor especificado, donde los tipos pueden ser float o double.
- public Float (String s). Construye un objeto Float inicializando el valor especificado por el parámetro del String.

Ahora se verán algunas instancias de las variables de la clase Float:

- public final static float MIN_VALUE. El mínimo valor posible para un float es (1.40129846432481707e-45).

- public final static float MAX_VALUE. El máximo valor posible para un float es (3.40282346638528860e+38).
- public final static float NEGATIVE_INFINITY. Negativo infinito.
- public final static float POSITIVE_INFINITY. Positivo infinito.
- public final static float NAN. No un número (esto es no es igual a ninguno, incluyéndose el mismo).

Algunos de los métodos de la clase Float son los siguientes:

- public static native String toString(float f). Regresa un String representando el valor del float especificado.
- public static native Float valueOf(String s). Regresa el valor del punto flotante representado por el String especificado.
- public boolean isNaN(). Regresa verdadero si este valor Float es un NaN (No un número).
- public boolean isInfinite(). Regresa verdadero si el valor de este Float es infinitamente grande.
- public String toString(). Regresa el String que representa al objeto Float.
- public int intValue(). Regresa el valor entero del objeto Float. (Hace una conversión a un entero).
- public long longValue(). Regresa el valor long del objeto Float.
- public float floatValue(). Regresa el valor de la variable float del objeto Float.
- public double doubleValue (). Regresa el valor double del objeto Float.
- public boolean equals(Object obj). Compara este objeto con obj, regresando verdadero si el valor o contenido de los objetos es el mismo.

1.5.4.5 Clase Boolean

Cada valor boolean tiene una clase Boolean. La clase Boolean está más limitada que las numéricas. Esta clase extiende de la clase Object.

Las variables de tipo Boolean se declaran de la siguiente manera:

```
boolean b;  /* tipo primitivo */      Boolean B;  /* tipo de referencia */
```

Los constructores de la clase Boolean son:

- public Boolean (boolean b). Construye un objeto Boolean inicializándolo con el valor boolean b especificado.
- public Boolean(String s). Construye un objeto Boolean inicializándolo con el valor especificado en el parámetro String.

Los valores que puede tomar esta variable son false (falso) o true (verdadero). Los métodos de la clase Boolean son:

- public boolean booleanValue(). Regresa el valor del Objeto Boolean como un boolean primitivo.
- public String toString(). Regresa el nuevo objeto String que representa el valor de este Boolean.
- public boolean equals(Object obj). Compara este objeto con el objeto especificado obj, regresa verdadero si el valor o contenido de los objetos es el mismo.

1.5.4.6 Clase String

Un objeto String representa un string que tiene un valor constante y que puede no ser cambiado después que ha sido creado. Un String representa una constante de un objeto. La clase String extiende de la clase Object.

Algunos de los constructores de la clase String son:

- public String(String value). Construye un nuevo String que es una copia del String especificado.
- public String(char value[]). Construye un nuevo String del cual su valor es el especificado por el arreglo de caracteres.
- public String(char value[], int offset, int count). Construye un nuevo String del cual su valor inicial es especificado por el subarreglo de caracteres. La longitud del nuevo String será contando los caracteres empezando del valor offset especificado.

La clase String cuenta con muchos métodos que permiten facilitar el uso de éstos, por ejemplo:

- public length(). Regresa la longitud del String.
- public char charAt(int index). Regresa el caracter en el índice especificado.

También cuenta con métodos de comparación como:

- public boolean equals(Object obj). Compara este String con el objeto especificado. Regresa verdadero si el objeto tiene la misma longitud y los mismos caracteres en la misma secuencia que este String.

- public boolean equalIgnoreCase(String str). Compara este String con otro String str. Regresa verdadero si el String str tiene la misma longitud y los mismos caracteres y la misma secuencia que este String, no importándole si están con mayúsculas o con minúsculas.
- public int compareTo(String str). Compara este String con otro String str. Regresa un entero si es menor que, igual que, o mayor que cero, dependerá del otro String si es menor que, igual que, o mayor que.
- public boolean startsWith(String prefix). Determina si el String empieza con el prefix.
- public String substring(int empieza, int termina). Regresa el substring de este String especificando dónde empieza y dónde termina el nuevo String.
- public String concat(String str). Crea un nuevo String, concatenando el String str al final del String que ya existía.

Ejemplo:

```
class stringPrueba{
   public static void main(String arg[]) {
      String x="abc";
      String y=new String("abc");
      String z="abc";
      if (x==y)
        System.out.println("x==y");
      else if (x.equals(y))
         System.out.println(" x igual que y");
      if (x == z)
         System.out.println("x == z ");
   }
}
```

El programa produce la siguiente salida:

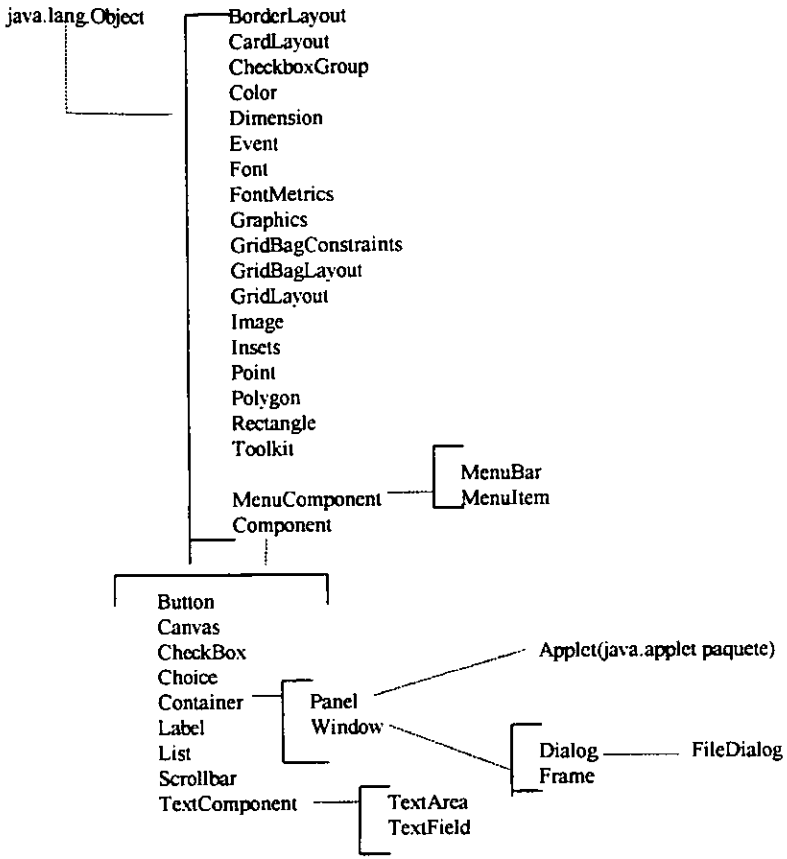
x igual a y
x=y

Métodos de conversión son:

- public String toLower case(). Crea un String convirtiendo todos los caracteres en minúsculas.
- public String toUpperCase(). Crea un String convirtiendo todos los caracteres en mayúsculas.
- public String toString(). Convierte este String a un String, regresando el mismo string.
- public static String valueOf(tipo x). Regresa un String que representa el valor del tipo x especificado, donde el tipo puede ser un int, float, double, char, boolean o un objeto.

1.5.5 Jerarquía de Clases del paquete java.awt

El paquete java.awt contiene clases que permiten crear interfaces gráficas. Una forma de ver las clases que contiene este paquete se muestra a continuación:



1.5.5.1 La Clase Container

Es importante saber que un Container puede contener Componentes e incluso otro Container.

Las ventanas o Paneles son sólo objetos que pueden contener otros componentes. Esta es la razón por la que se pueden ver botones, listas, scrollbars, y otros componentes en los Frames y paneles.

Cuando se tenga que crear un componente se usa el constructor y para adherirlo al Frame o panel se utiliza el método add(). Para poder utilizar estas clases se necesita importar java.awt.*;.

Para crear un Frame, su constructor es Frame(String) que crea un Frame que es invisible, el String es el título que este va a tener. Para poderlo mostrar se necesita usar el método show(), que hace que el Frame sea visible.

Para poder construir un panel se utiliza el constructor Panel(). Este panel es creado y puede ser añadido a la ventana o Frame, con el método add.

Un ejemplo de cómo crear un Frame, que contenga un panel amarillo se muestra a continuación:

```
import java.awt.*;
public class prueba extends Panel{
  public static void main (String arg[])}
     Frame fr = new Frame ("Frame con un panel");
     Panel pan = new Panel();
     fr.resize(200,200);  // la dimensión del frame es de 200x200
     fr.setBackground(Color.blue);   // el fondo  del frame es de color azul
     fr.setLayout(null);    // no tiene definido ningún tipo de arreglo para la pantalla
     pan.resize(100,100);  // el margen del panel es de 100 x 100
     pan.setBackground(Color.yellow);   //el color del panel es amarillo
      fr.add(pan);        // se está añadiendo el panel al frame
      fr.show();
  }
}
```

Para añadir botones a cualquier Container se declara:

```
Button btn = new Button(" Ok" ); //en donde Ok es la etiqueta que va a tener el
                                 //botón
add( btn );
```

1.5.5.2 La Clase LayoutManager

El ambiente de trabajo, o mejor dicho, la manera en la que se quieren acomodar los componentes se determina con los Layout:

- FlowLayout. Este layout es el que se tiene por default, en los paneles, que los va acomodando horizontalmente.
- BorderLayout. Este layout es un poco más complejo, contiene cinco distintas áreas para acomodarlos que son North(Norte), South(Sur), East(Este), West(Oeste) y Center(Centro). Este layout está por default en los Frames y Dialogs. Un pequeño ejemplo es el siguiente:

```
Frame fr =new Frame("BorderLayout"); // crea el Frame
Button botonNorte =new Button("Norte"); // crea el botón con la etiqueta Norte
fr.add("North",botonNorte); // adhiere el botonNorte en la posición norte del Frame
fr.show(); // muestra el Frame
```

- GridLayout. Este layout da un poco más de flexibilidad para poder acomodar los componentes ya que uno le indica el número de columnas y renglones que necesita, así como se puede determinar el largo y ancho de las mismas:

```
setLayout( new GridLayput(3,2)); // esto le indica que el layout es de tipo GridLayout
                                 // con 3 renglones y 2 columnas
```

1.5.5.3 La Clase Event

También se pueden detectar los eventos, para poder realizar alguna acción, es decir, que si se presiona, por ejemplo un botón con etiqueta Ok, realice una suma.

Los eventos que pueden ser detectados son:

Windows, Ya sea que la ventana se destruya, se haga pequeña, se haga grande o que se mueva.

Mouse, Detecta si el mouse se movió, si se apretó el botón derecho o izquierdo del mouse, etc.

KeyBoard, Si alguna Tecla se apretó como la de Enter, Insert, Inicio, ESC, etc.

Scrolling Si las flechas de la barra de scroll se presionaron.

1.5.5.4 Principales Componentes

Existen además otros tipos de botones como el Choice, Checkbox CheckboxGroup.

Para crear *TextField*, se necesita el constructor TextField;

```
TextField tx = new TextField();  // no se le especifica ni tamaño ni contenido.
TextField tx = new TextField(20); // se le indica que va a ser de  20 columnas.
TextField tx = new TextField("Hola"); // el texto va a desplegar Hola.
TextField tx = new TextField(Hola",20); //el texto es predefinido y con longitud de 20
                                        //columnas.
```

La creación de etiquetas se hace con el constructor *Label*:

```
Label lb=new Label("Hola");
```

La creación de listas se hace con el constructor *List*:

```
List ls= new List (4,false) // 4 renglones visibles y sólo se puede  seleccionar un
                            //elemento de la lista
```

Para añadir elementos en la lista se utiliza la siguiente instrucción:

```
ls.addItem("Mercurio");
```

TextArea necesita un constructor TextArea:

```
TextArea tx= new TextArea("Hola", 5,40) //que tiene texto definido, con 5 renglones y
                                        //40 columnas.
```

Para crear un *Scrollbar*:

```
Scrollbar rango=new Scrollbar(Scrollbar.HORIZONTAL, 0,64,0,255); /* que es un
```

scrollbar horizontal, la barrita que tiene en medio es de 64 unidades de longitud y el scrollbar puede desplazar de 0 a 255 unidades. */

Estos y otros componentes se pueden encontrar en la documentación en línea para JAVA que puede ser localizada en Internet en la siguiente dirección: http://www.java.sun.com/documentation/ y que puede ser bajada a disco.

Sólo se mencionan y explican estos componentes puesto que son los que se utilizarán más adelante en el Capítulo III de este trabajo de Tesina que trata sobre una aplicación hecha en Java para una aseguradora y que además utiliza tarjetas inteligentes.

Es importante que se haya entendido cómo crear un programa en Java, puesto que estos mismos principios se utilizarán en el siguiente capítulo cuando se hable de Tarjetas Inteligentes y sobre cómo hacer programas para éstas.

CAPÍTULO II

TARJETAS INTELIGENTES

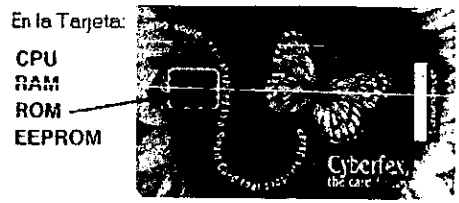
2.1 ¿Qué es una Tarjeta Inteligente?

Una tarjeta inteligente es usualmente del mismo tamaño que una tarjeta de crédito convencional, pero incorpora un pequeño contacto de metal dorado (microchip) o un módulo sobre la cara de la tarjeta, y que al igual que las tarjetas de banda magnética se necesita de un lector para poder accesar los datos contenidos en ella. La siguiente figura muestra un lector de tarjetas inteligentes.

FIGURA 2.1 Lector de tarjetas inteligentes

Este lector que se muestra en la figura 2.1 es uno de tantos que se pueden encontrar en el mercado, ya que dependiendo del tipo de tarjeta será el tipo de lector que se pueda utilizar. En la figura 2.2 se tiene un tipo de tarjeta inteligente llamado JavaCard, el cual es uno de entre cientos de tipos y diseños de tarjetas inteligentes existentes. Los tipos de tarjetas inteligentes se verán en el siguiente tema para que quede mejor entendido el concepto de estas.

FIGURA 2.2 Tarjeta inteligente



Los microchips no sólo tienen más poder, ellos también son de menor tamaño. El tamaño de los microchips ha hecho posible ponerlos dentro de las aplicaciones de tarjetas de crédito. Los chips permiten 100 veces más capacidad de almacenaje que una tarjeta de banda magnética y además permiten arriba de 1 millón de ciclos de escritura a ser realizados.

2.2 Tipos de Tarjetas

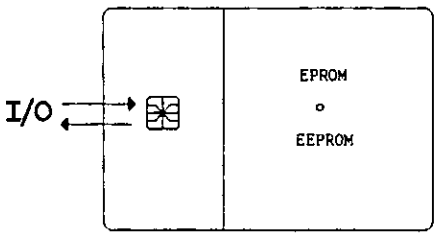
Hay tres tipos básicos de chips (aunque no se tiene una clasificación generalizada) disponibles para las tarjetas inteligentes:

1.- Chips con memoria de acceso aleatorio (RAM).
2.- Chips con memoria de seguridad lógica.
3.- Chips con procesador.

1.- Los Chips con memoria EPROM (Memoria programable y borrable de sólo lectura) son programados una vez, y pueden entonces ser leídos sin restricciones. Los chips con memoria EPROM pueden ser programados y leídos tan frecuentemente como se requiera. Tales tarjetas son usadas en lugar de tarjetas de banda magnética, puesto que son más confiables y ofrecen mucha más memoria.

No todos los tipos de memoria permiten el borrado de datos, el cual es un requerimiento básico para muchas aplicaciones. La escritura se refiere usualmente al cambio de un bit 0 a un bit 1 o viceversa, mientras que actualizar es borrar el contenido de las celdas de memoria seguidos por la escritura. La figura 2.3 ayuda a visualizar las características de esta tarjeta de memoria.

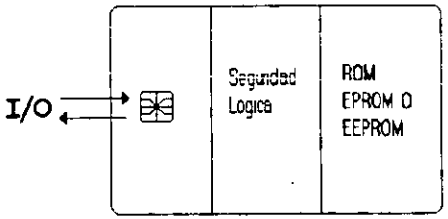
FIGURA 2.3 Tarjeta de memoria o de chips



2.- Los chips con memoria de seguridad lógica permiten la introducción de atributos seguros para lectura y escritura de datos. Una zona de memoria puede ser secreta (el dato es usado por la tarjeta solamente para propósitos internos), pública o sensitiva. La última significa que puede ser accesible solamente después de presentar un correcto identificador de usuario. Este es en muchos casos un Número de Identificación Personal (NIP o PIN por sus siglas en Inglés) consistiendo de 4 a 8 dígitos.

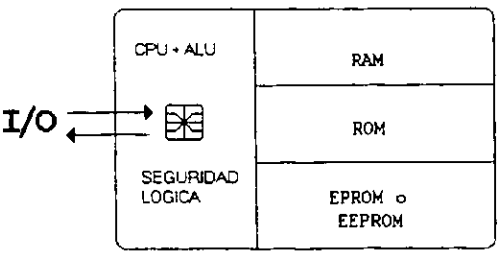
El NIP está protegido contra ataques de prueba y error, por un "contador de falsa presentación". Después de un número especificado de falsas entradas consecutivas, la seguridad lógica bloquea los datos no públicos contra más accesos. La nueva generación de telecomunicación chip (para aplicaciones de tarjetas telefónicas) también incluye algoritmos de hardware para un mecanismo de desafío de respuesta para la autentificación de la tarjeta por el sistema, incrementando la seguridad contra copias. Estas características se pueden ver en la figura 2.4.

FIGURA 2.4 Tarjeta con Seguridad Lógica



3.- Los chips con procesador son una completa unidad de procesamiento central (CPU), con memoria para sistema operativo (ROM), una memoria principal y un sector de memoria para los datos de la aplicación (EEPROM). Esta inteligencia activa soporta la realización de varias y diferentes aplicaciones en un chip. Un particular alto nivel de seguridad puede ser llevado a cabo por medio de métodos criptográficos. La mayor funcionalidad de los chips con procesador es realizar el incremento de complejidad del diseño del chip sobre largas áreas del chip en comparación con las anteriores tarjetas. El tiempo de programación durante la producción de una tarjeta inteligente es una razón del elevado precio sobre las tarjetas anteriores. Lo que se puede ver en la figura 2.5.

FIGURA 2.5. Distribución de componentes de una tarjeta



2.2.1 Niveles de inter-operabilidad

El éxito de las Tarjetas inteligentes se ha dado por los tres niveles de Inter-operabilidad, es decir por las tres características principales que componen la arquitectura de las tarjetas. Estos tres niveles son los siguientes:

1.- Físicos
2.- Plataforma
3.- Aplicaciones.

1.- Requiere del estándar de ISO7816 para los requerimientos físicos de los contactos, de las terminales y de la zona de los contactos.

2.- Los estándares de la plataforma permiten a la tarjeta compartir un Sistema Operativo con un lenguaje y una estructura de archivo similar. Estos permiten aplicaciones de una gran variedad que se pueden correr sobre la tarjeta inteligente y sobre el mismo lector de la tarjeta.

3.- Dentro de algunas industrias ya hay aplicaciones funcionando, más particularmente para tarjetas de crédito inteligente. Otras aplicaciones tales como monedero electrónico tienen un número finito de estándares (para aplicaciones específicas) tales como MONDEX, VISACASH y Proton.

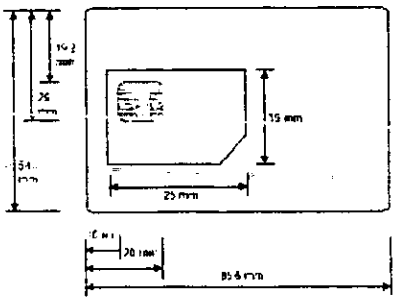
2.3 Estándares

El estándar ISO 7816 es un conjunto de 8 documentos que describen los aspectos físicos de los circuitos integrados de las tarjetas. Las partes de este estándar están resumidos, a continuación:

2.3.1 Características Físicas

El estándar de la tarjeta o ID – 1 es del tamaño de una tarjeta de crédito. La figura 2.6 da las dimensiones aproximadas de esta tarjeta y localización de los contactos así como la altura y ancho La tarjeta ID – 000, con una altura de 15 mm y 25 mm de ancho ha sido primero especificada por el Instituto Europeo de Estándares de Telecomunicaciones de GSM (Sistema Global de Comunicaciones Móviles), donde más de la mitad del número de tarjetas tiene este formato. Estas son principalmente usadas en teléfonos móviles, los cuales son muy pequeños para aceptar una tarjeta ID – 1.

FIGURA 2.6 Tamaños de las tarjetas



Un tercer tipo, las así llamadas "Mini Card" o tarjeta ID – 00 es de 66 mm de ancho y 33mm de alto. Tiene la misma localización de los contactos con respecto a la parte más alta y a la izquierda de la tarjeta ID – 1. Esta permite el mismo lector de tarjeta, si así es diseñada, para aceptar cualquiera de ellas. Ambos formatos pueden ser obtenidos de una tarjeta ID – 1 cortando el exceso de plástico.

Además de los 8 contactos, la tarjeta puede ser equipada con una banda magnética y un grabado. El grabado y la banda magnética estarán al lado opuesto del chip, mientras, la versión actual de ISO/7816 – 2 permite contactos y banda magnética del mismo lado. Esta posibilidad ha sido excluida en la versión del estándar ISO/7816 el cual se espera que se publique a principios del año 2000.

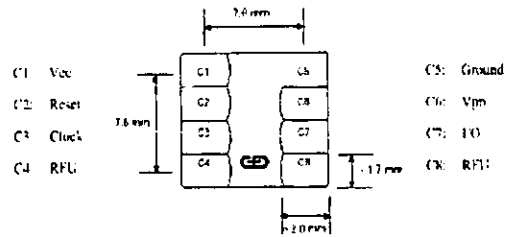
La figura 2.7 muestra la única posibilidad para una tarjeta la cual provee las tres interfaces.

FIGURA 2.7 Interfaz de la tarjeta inteligente

2.3.2 Dimensión y localización de los contactos

En la figura 2.8 se muestran los contactos del chip. Además se describen las características electrónicas y los requerimientos de corriente.

FIGURA 2.8 Contactos del chip



C1 (Vcc): Este contacto es el alimentador de voltaje del lector de la tarjeta hacia el chip. Es usualmente de 3v a 5v.

C2 (Reset): Este es el pin que recibe la señal de "reset" del lector de la tarjeta. La tarjeta responderá inmediatamente a esta señal.

Podríamos definir el "reset" como el montaje original de los datos de la tarjeta, conocido en el lenguaje de las tarjetas inteligentes como ATR (Answer-To-Reset). El uso primario del ATR es indicarle al lector el tipo de tarjeta que ha sido insertada en el lector.

C3 (Clock): Es para la señal del reloj externo (de 3.5Mhz a 5Mhz) del lector que opera el CPU de la tarjeta.

C5 (Ground): Es el regreso de corriente o "tierra".

C6 (Vpp): Este solía ser el pin de programación de voltaje para programar las EEPROM para algunas tarjetas. Ya que las actuales tarjetas tienen un circuito de control de voltaje interno, ha caído en desuso.

C7 (I/O): Es la entrada serial y el pin de salida. El flujo de datos entre la tarjeta y el lector va por este pin. Como la tarjeta ocupa solo un cable para comunicarse con el lector, la información puede solamente fluir en una sola dirección a la vez.

Los restantes pines (C4 y C8) están reservados para usos futuros.

2.3.3 Instrucciones de la Tarjeta

Las tarjetas se comunican con el lector vía bloque de comandos llamados APDU's (Application Protocol Data Units, por sus siglas en inglés). Los APDU's se definen como el conjunto básico de APDU's que la tarjeta debe entender. Los APDU's consisten de los siguientes campos:

Formato de Comandos APDU

CLA	INS	P1	P2	Lc	Data	Le

CLA: clase de instrucción.

INS: Número de la instrucción.

P1: Parámetro 1.

P2: Parámetro 2.

Lc: es un Parámetro que indica si va a regresar información

Data: Datos a enviar.

Le: es la respuesta de la operación

Formato de Respuesta APDU

Data	SW1	SW2

Data: Información regresada.

SW1: Palabra de estado de la respuesta

SW2: Palabra de estado de la respuesta

2.3.4 Protocolo de Transmisión

Para el intercambio de datos entre la tarjeta inteligente y el dispositivo de interfaz se han estandarizados dos protocolos. Están denotados por T=0 y T=1, respectivamente. Ambos son protocolos asíncronos "half duplex". La principal diferencia entre los dos es la manipulación de datos, donde el T=0 es una transmisión byte por byte, requiriendo poca memoria, y T=1 es una transmisión por bloques de bytes.

Verificar la paridad de un byte inmediatamente después de haberlo recibido y solicitar la retransmisión es la única corrección de error. No es posible transmitir datos en ambas direcciones. Los datos tienen que ser enviados uno a uno hasta que el buffer pueda arreglárselas con los bytes restantes. Estos pueden entonces ser solicitados usando un "acknowledgement" (confirmador) para el último byte recibido.

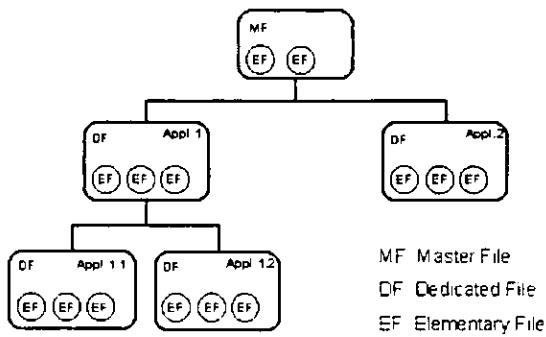
2.3.5 Tipos de Archivos

El sistema operativo de las tarjetas inteligentes administra archivos en directorios – similar a los sistemas operativos de las PC convencionales –. Se distinguen los siguientes componentes dentro de la organización de los archivos los cuales pueden visualizarse teniendo tres estructuras:

- Archivo Maestro (MF): la raíz del sistema de archivos.
- Archivos dedicados (DF): un DF contiene subestructuras y la aplicación actual con sus archivos elementales (los DF son llamados también directorios).
- Archivos elementales (EF): un EF contiene información de las aplicaciones o datos.

En la figura 2.9 se muestra la estructura de "File System" que se tiene, dadas las anteriores descripciones de archivos.

FIGURA 2.9 Estructura de File System



El MF es usualmente seleccionado implícitamente después que la tarjeta ha sido montada de nuevo. Puede contener también una aplicación sobre tarjetas mono–aplicación. En el caso de tarjetas multi–aplicación cada aplicación está contenida en archivos dedicados separados o en otros subdirectorios. Una aplicación es seleccionada usando un identificador de aplicación (AID), el cual puede estar registrado en los niveles nacionales e internacionales de ISO/IEC, o por un identificador de archivos que consiste de dos bytes. Los identificadores de archivos tienen la ventaja que el dispositivo de interfaz no tiene que conocer el identificador de archivo para la aplicación y ese registro es único. Varias aplicaciones pueden estar fácilmente combinadas en la tarjeta. En el caso de identificadores de archivos una colisión puede aumentar si dos aplicaciones usan el mismo identificador y están combinadas en la tarjeta. Los datos, los cuales son usados por todas las aplicaciones

en la tarjeta (por ejemplo, número serial, llaves, NIP) y datos concernientes a la administración del ciclo de vida de la tarjeta, están almacenados en el archivo maestro (MF).

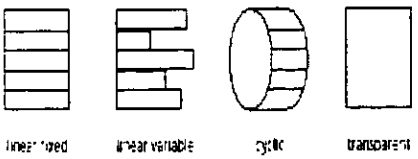
Esta información es usada por el sistema operativo para la creación de una nueva aplicación. El control de información de aplicaciones específicas y los archivos son almacenados en un DF que contiene la aplicación. Ellos están separados lógicamente y, en algunos casos, físicamente de otras aplicaciones contenidas en diferentes DF's. La correcta administración para los DF's ordenados en una estructura "vertical" es siempre ejecutada por el DF padre, es decir, el DF que reside un nivel arriba. Un DF puede administrar varios archivos dedicados los cuales están en los niveles de abajo.

2.3.6 Estructura de Archivos

La estructura de archivos de un EF depende de su uso. Los siguientes cuatro tipos están definidos en ISO 7816-4 y se muestran en la figura 2.10:

- Los Archivos transparentes (Transparent) consisten de una secuencia de bytes teniendo una estructura amorfa.
- Los Archivos lineales fijos (Linear Fixed) consisten de registros teniendo la misma longitud.
- Los Archivos lineales variables (Linear Variable) consisten de registros de longitud variable.
- Los Archivos cíclicos (Cyclic) tienen registros de longitud fija los cuales están organizados en una estructura de anillo, la última entrada será sobrescrita por la entrada a ser almacenada.

FIGURA 2.10 Estructuras de archivo en ISO 7816-4



Las cuatro estructuras básicas pueden ser extendidas por otras estructuras de archivos. Algunos ejemplos son los archivos de bases de datos, archivos de llaves públicas internas, entre otros.

La definición de categorías, los tipos y las posibilidades de acceso (leer, escribir, actualizar) con sus respectivas condiciones de acceso (nunca, siempre, NIP, especial, . . .) están almacenadas en la cabecera de cada archivo.

2.3.7 Arquitectura de seguridad

Los sistemas operativos muy frecuentemente administran aplicaciones con altos requerimientos de seguridad. La seguridad de una tarjeta inteligente es una combinación de la seguridad del chip (hardware) y el sistema operativo (software). Para obtener una óptima seguridad la programación de un sistema operativo requiere de conocimientos extensivos de propiedades del hardware en particular con respecto a características eléctricas, detectores, interrupciones y otras características las cuales pueden influir en la seguridad. Desde el punto de vista de software se puede distinguir entre seguridad funcional y seguridad contra manipulaciones.

La seguridad funcional puede ser garantizada por:

- Protocolo de transporte.
- Intérprete de comandos.
- Organización de archivos, estructura de archivos.
- Funciones.
- Separación de capas.
- Detección de errores y funciones de corrección.

Seguridad contra manipulación puede ser obtenida por implementación:

- Mensajes de seguridad.
- Identificación y autentificación.
- Firmas digitales.
- Generación de números aleatorios.

Los tipos de archivos pertenecientes a la tarjeta pueden estar protegidos también por medio de *Atributos de Acceso* y por *Condiciones de Acceso*. Por ejemplo un atributo puede ser el de lectura, y una condición que esté insertada la tarjeta. Esto significa que para poder tener acceso al archivo será necesario introducir un password o una llave de entrada.

Los mecanismos de seguridad básica definidos en los estándares de la ISO7816-4 son los siguientes:

- Autentificación mediante password.

  Los usuarios deberán proveer un password antes de accesar a lo permitido.

- Autentificación mediante llave.

  Los usuarios deberán proveer una llave que ya está almacenada en la tarjeta antes de accesar a lo permitido.

- Autentificación de datos.

  La tarjeta le adjunta un código a los datos, para que el usuario se asegure que la tarjeta no sea falsa.

- Encripción de datos.

  La tarjeta encripta los datos para que su integridad sea mantenida.

Cada Sistema Operativo de las tarjetas tiene implementada esta seguridad, usando la combinación de atributos con condiciones de acceso. Esta opera en los niveles de archivos EF, MF, DF. Generalmente en el nivel de MF y DF se tiene un alto nivel de seguridad, así que solamente si es provista la llave de estos niveles o archivos se podrá manipular la información.

Los atributos de acceso permitidos son:

- ALW (Siempre): Como su nombre lo dice tiene acceso libre a los archivos para leer y actualizar.

- NEV (Nunca): Es el más alto nivel de protección ya que no se tiene permiso para accesar.

- CHV1 (Verificación del Titular de la Tarjeta): La ventaja de una tarjeta inteligente sobre una de banda magnética es que el password puede ser almacenado en la tarjeta de una forma que no pueda ser leído desde afuera. Usualmente la tarjeta tiene dos passwords, CHV1 y CHV2, donde el segundo es para propósitos administrativos.

- CHV2 (Verificación del Titular de la Tarjeta).

- PRO (Protegido): sirve para verificar que los mensajes entre la tarjeta y la aplicación huésped no estén alterados. Un Código de autentificación de mensajes (MAC, por sus siglas en inglés) es agregado al mensaje, si esta condición de acceso es especificada.

- AUT (Autentificación): Hay dos métodos de autentificación:
  Interno: La aplicación debe de informar a la tarjeta, que está autorizada para leer y actualizar datos en la misma, es decir ambos deben tener las mimas llaves.
  Externo: La tarjeta debe informar a la aplicación que está autorizada para enviar datos a la aplicación.

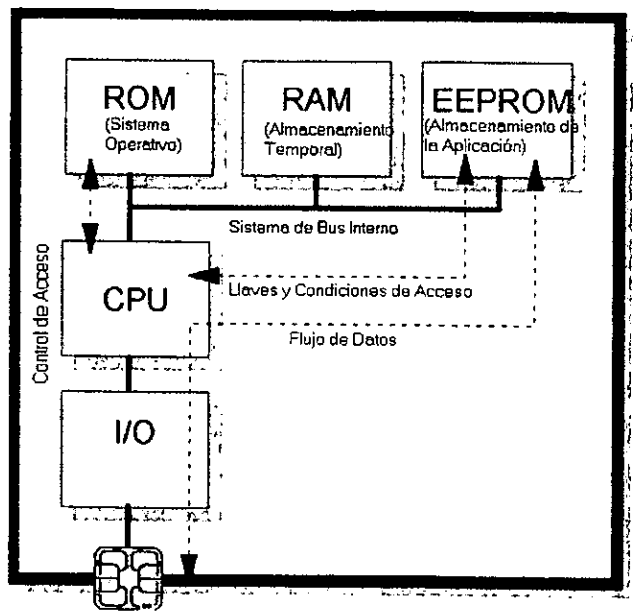
- ENC (Encriptado): es similar al PRO, a excepción que este código está encriptado.

- Combinaciones con CHV1 o CHV2:
  - CHV1/PRO
  - CHV1/AUT
  - CHV1/ENC

2.4 Microprocesador de la Tarjeta Inteligente

Hoy en día los microprocesadores son simples chips, lo cual significa que todas las partes mostradas en la figura 2.11 (abajo), están integradas dentro de una sencilla pieza de silicón. Esto tiene algunas ventajas desde el punto de vista integridad y seguridad. Conectar el chip al área de contacto y colocar este módulo dentro del plástico es una profesión especializada.

FIGURA 2.11 Partes de un micro-procesador



CPU

El CPU es usualmente un microprocesador de 8 bits con un bus de direcciones de 16 bits. Esto permite a esta tarjeta de chip direccionar a un máximo de 64 bytes.

ROM

La Memoria de solo lectura (ROM) es una máscara programada. El código ROM debería solamente contener programas y datos los cuales no sean específicos para una tarjeta pero sean lo mismo para un gran número de tarjetas y las cuales sean constantes durante su vida. Un cambio del código ROM requiere una nueva máscara. El código ROM contiene típicamente el sistema operativo de la tarjeta, el protocolo de transmisión y comandos, los algoritmos de seguridad y el software para las aplicaciones. Las llaves específicas de la tarjeta necesitan para su ejecución las funciones de seguridad que están contenidas en la memoria no – volátil.

RAM

La Memoria de acceso aleatorio (RAM) es una memoria volátil, el contenido de la cual se pierde cuando su poder de alimentación es apagado. El tamaño de esta memoria es de alrededor de 256 Kb. Esta es usada por el CPU como un buffer para almacenar transmisión de datos y como un muy rápido acceso a memoria para los resultados intermedios producidos durante la ejecución de un algoritmo. Leer o escribir un byte toma pocos segundos, siendo 1000 veces más rápido que escribir un byte dentro de la EPROM.

EEPROM

La memoria no – volátil programable de las tarjetas de hoy en día consiste de EEPROM (Memoria de sólo lectura, borrable y programable eléctricamente). Este tipo de memoria permite un mínimo de 100,000 ciclos de actualizaciones. Ellos derivan internamente la programación de voltaje requerido, que es de aproximadamente 18v de la alimentación de voltaje Vcc. La primera generación de tarjetas tenía EPROM (Eraseble Programmable Read Only Memory) con una fuente externa para la programación del voltaje Vpp. Como el contenido de la EPROM puede solo ser borrado por luz UV, cada celda puede ser programada solo una vez por el CPU. El uso de las tarjetas con EPROM fue así limitado por las aplicaciones que no requieren actualizaciones frecuentes de memoria. Una importante condición de límite es la superficie del chip el cual no debe exceder a 25 mm^2.

I/O

Es el puerto de entrada y salida de la tarjeta usado para transferir datos bit por bit.

CO – PROCESADOR

Las tarjetas más avanzadas están provistas de un co–procesador el cual ejecuta las operaciones sobre enteros, cuando se manejan procedimientos de encripción para firmas digitales.

2.5 Sistema operativo de las Tarjetas Inteligentes

Los sistemas operativos son usados sobre diferentes plataformas de hardware digitales – una tarjeta inteligente, calculadora de bolsillo, PC o mainframes -. Hablando en términos generales el término sistema operativo, para el cual no hay definición general, se refiere a un grupo de programas de sistema requerido para operar una computadora.

El sistema operativo provee funciones definidas para el usuario, quien no necesita conocer nada acerca del hardware de la computadora. El usuario puede correr y programar aplicaciones casi completamente independientes del hardware. El sistema operativo se encarga de controlar y organizar los medios de memoria (RAM, disco duro, CD's, etc.) y de procesos de programación. Esto permite a la manufactura del hardware incorporar muchos desarrollos tecnológicos sin necesitar un cambio de funciones del sistema operativo. El sistema operativo de las tarjetas inteligentes y sus tareas básicas pueden ser caracterizados como sigue:

- El mini sistema operativo requiere capacidad de memoria de pocos bytes.
- Administración del hardware de seguridad basado sobre microcontroladores de chips.
- Simple sistema de procesamiento.
- Interfaces a la máquina mediante interfaz serial.
- Soporte de métodos criptográficos para autentificar componentes de sistema.
- Seguridad en almacenamiento de datos y provisión de seguridad para las aplicaciones.

El sistema operativo y el hardware de las tarjetas inteligentes tienen que protegerse ellos mismos contra las tres siguientes amenazas para la duración de su ciclo de vida:

- Pérdida de confidencialidad, burlando o desautorizando información referente a los programas y datos tales como llaves, PIN's y datos de usuario.
- Pérdida de integridad, manipulando o desautorizando modificación de información tal como números de identificación y contadores de propósitos electrónicos.
- Pérdida de disponibilidad desautorizada sujetando información o funciones de sistema.

2.5.1 Ciclo de vida de las Tarjetas

El sistema operativo contribuye considerablemente a garantizar ciclos de vida de las tarjetas. El ciclo de vida es dividido dentro de las siguientes cinco fases de acuerdo a ISO 10202-1:

- Fase 1: Manufactura del chip y la tarjeta.
- Desarrollo del sistema operativo y manufactura del transporte del chip.
- Implementación del sistema operativo, usualmente como una máscara de ROM
- Producción de chip y manufactura del transporte del chip.
- Fase 2: Preparación de la tarjeta.
- Inicialización y prepersonalización de la tarjeta, es decir, cargar constantes y datos de liberación de sistema.
- Fase 3: Preparación de la aplicación.
- Asignación, personalización y activación de una o varias aplicaciones.
- Fase 4: Fases de aplicación.
- Usando funciones de tarjetas globales y acceso a aplicación.
- Usando manejo de funciones o aplicaciones.

- Fase 5: Terminación de uso.
- Borrar llaves y completar aplicación.

2.6 ¿Cómo interactúa Java con la Tarjetas Inteligentes?

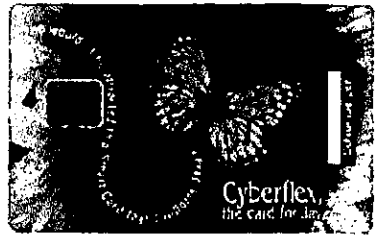
Ya se ha visto cómo es interna y físicamente una tarjeta inteligente, pero ahora se verá cómo y para qué podemos utilizarla, pero lo más importante es que se enfocará, ahora sí, a la tarjeta que soporta Java. Se verá como se programan, además de que se dará una breve introducción a la documentación de las clases que permiten accesar a los datos contenidos en la tarjeta. Existen compañías que producen tarjetas inteligentes, pero sobre la cual se basa este trabajo es la tarjeta de Schlumberger, porque es sobre la que se tiene más contacto y sobre la que se tiene más conocimiento, además porque con esta compañía se mantiene una mejor relación pudiendo así conseguir información más fácilmente e incluso el equipo. Por tal motivo los temas que abajo se mencionarán estarán enfocados a los productos de dicha compañía (lector, tarjeta, lenguaje de programación).

2.6.1 JavaCard, la tarjeta Inteligente para Java.

Cyberflex™ 2.0 Multi 8K. es la última realización de tarjetas inteligentes de Schlumberger que corre programas escritos en Java.

La tarjeta Cyberflex es muy parecida a una tarjeta de crédito pero en realidad es una pequeña computadora. Las tarjetas contienen un microprocesador como se puede ver en la figura 2.12, que proporciona capacidad de procesos y memoria para almacenar instrucciones y datos.

FIGURA 2.12 Tarjeta Cyberflex



Las tarjetas pueden ser usadas para almacenar y actualizar una gran cantidad de información y datos personales e incluso valores monetarios. Las tarjetas son ideales para un acceso seguro a Internet, compras, teléfonos digitales portátiles, y para programas y aplicaciones útiles para la salud. Las tarjetas inteligentes aportan nuevos servicios, tan buenos como el incremento de la seguridad, portabilidad y comodidad de aplicaciones. Cyberflex aumenta en gran medida el potencial de las tarjetas inteligentes.

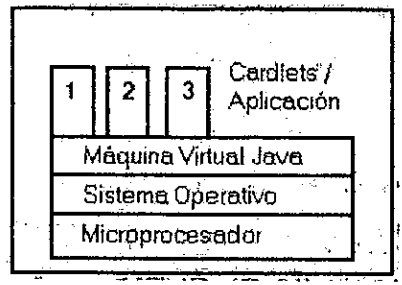
Cyberflex es una tarjeta inteligente de Java (JavaCard™). Contiene la Java Virtual Machine (JVM) y Sistema operativo.

Cyberflex 2.0 Multi 8k. la última realización de JavaCard, incrementa grandemente el espacio disponible para la aplicación. Además los programas en Java pueden accesar ahora a la interfaz "Default Loader" que reside en la tarjeta. Este acceso de conexión de las tarjetas es el proceso compatible por default, permite un uso más eficiente del espacio disponible en las tarjetas. Cyberflex 2.0 Multi 8k. tiene compatibilidad con versiones anteriores de Cyberflex.

2.6.2 Arquitectura Cyberflex

La figura 2.13 muestra la arquitectura de una tarjeta inteligente Cyberflex.

FIGURA 2.13 Arquitectura de la tarjeta inteligente Cyberflex



Estas son las tres capas de software en la arquitectura Cyberflex.

1. Cardlets™ que son cargados a la tarjeta. El número de cardlets que puede haber en una tarjeta está limitado solo por el espacio.

2. La JVM se incluye en cada tarjeta Cyberflex. La JVM es un intérprete que traslada el código byte de Java en instrucciones que el microprocesador de la tarjeta inteligente pueda entender. La JVM aporta a la tarjeta inteligente toda la flexibilidad y portabilidad provista por el lenguaje Java.

3. El Sistema Operativo de Propósito General (GPOS) está incluido en cada tarjeta Cyberflex. Esta herramienta de operación de la tarjeta está basada en los estándares de producción de la ISO. El estándar es llamado ISO 7816. El sistema operativo realiza ciertas partes del estándar, ya que proporciona administración de mensajes, administración de archivos, administración de seguridad y utilerías.

2.6.3 Características de Cyberflex

Las JavaCard Cyberflex dan todas las ventajas del lenguaje Java junto con las tarjetas inteligentes. Esta proporciona las siguientes características.

JavaCard

La Cyberflex ofrece las siguientes ventajas:

1. Un lenguaje de programación estándar. Quien sepa cómo hacer un programa en Java puede hacer una aplicación en las tarjetas inteligentes y cargarlas en las tarjetas Cyberflex.

2. Seguridad. Java es bien conocido como un lenguaje de programación seguro, ya que Java no hace referencias a programas desarrollados en rutinas de asignación de memoria.

3. Aplicaciones múltiples o cardlets en una tarjeta. La arquitectura de las Cyberflex y las características de seguridad del lenguaje Java hacen esto posible para múltiples cardlets que están en la tarjeta. Estos cardlets pueden ser servicios que funcionan juntos y son del mismo proveedor (por ejemplo, crédito, débito y servicios de efectivo en una compañía de tarjetas de crédito) o podrían estar relacionados con aplicaciones de diferentes proveedores.

   Como ya se había mencionado anteriormente el número de cardlets está únicamente limitado por el espacio en la tarjeta.

4. Todos los beneficios de la programación orientada a objetos están disponibles para los programadores Cyberflex: por ejemplo, código de re - uso, diseño de dibujos y estructura superior.

5. Multiplataforma. Puesto que los programas en Java en las tarjetas inteligentes son portátiles a través de los diferentes chips de arquitectura, el costo de desarrollo y mantenimiento es menor.

Integridad de datos y seguridad

La seguridad de Cyberflex está tanto en el software como en el hardware. La integridad de los datos y la seguridad están dadas mediante las características de Java y el GPOS.

Integridad y seguridad del ambiente Java

La integridad y seguridad de Java son muy reconocidas. Las siguientes son características proporcionadas por el programa e integridad de datos y seguridad de programas *Maliciosos*:

- El compilador Java proporciona una extensa verificación de errores graves cuando el programa es compilado. Por ejemplo, todos los métodos de referencia y variables son examinados para asegurarse que los objetos son del mismo tipo. El compilador también examina para asegurar que el programa no accese ninguna variable no inicializada.

- Todos los métodos de acceso y variables de instancia en una archivo Java *class* son mediante modificadores de acceso. Estos modificadores definen el nivel de control de acceso de cada método. Se puede declarar un método que sea *public* (sin limitaciones), *protected* (accesible para métodos en la misma subclase o paquete) o *private* (no accesible para otras clases). Si no se hace la declaración, el permiso por *default* del método es accesado de cualquier clase en el mismo paquete.

- Los tipos básicos de Java y operadores están bien definidos. Todos los tipos primarios tienen un tamaño específico, y todas las operaciones se realizan en un orden señalado.

- Los programas maliciosos no pueden "forzar" indicadores de memoria porque no son indicadores que puedan ser accesados por programadores o usuarios.
  Los programas maliciosos son impedimentos para fisgones alrededor de Java porque los valores de las variables locales no están disponibles después de invocar cada método. Un método no puede accesar recursos.

- Aunque Java libera automáticamente las localidades de memoria (recolector de basura), las Java Cards no. La basura se genera cuando las instancias a clases y arreglos son inicializados en la pila, pero no se vuelve a hacer referencia a ellas. Las localidades típicas de Java, las cuales son partes de memoria en uso, automáticamente limpian las partes que no han sido usadas.
  Con las aplicaciones en las tarjetas inteligentes, este tipo de administración de memoria no es necesario. Los tiempos de transacción en las tarjetas inteligentes son tan pequeños que la basura no es problema.

Seguridad en Hardware

El hardware en las tarjetas inteligentes es seguro porque las tarjetas son resistentes. Es muy difícil tener por una parte la tarjeta y leer el código en el chip.

Capacidad del núcleo

Cyberflex proporciona la capacidad de las tarjetas Java y el soporte y características que requieren las tarjetas inteligentes de hoy.

Las aplicaciones de las tarjetas inteligentes desarrollan típicamente las tareas de:

- Autentificación
- Lectura, escritura y actualización del acceso de datos.
- Fijación de un punto aritmético.

El núcleo de las tarjetas incluye:

- Datos de tipo boolean, byte y short.
- Relación de flujo de control.
- Modificadores de acceso y operadores.
- Arreglos unidimensionales de soporte de datos

Las siguientes características de Java no están incluidas:

- Caracteres Unicode.
- Enteros de 32 y 64 bits.
- Datos de tipo double y float.

- Arreglos multidimensionales.
- Arreglos unidimensionales de datos no soportados.
- Arreglos de objetos
- Excepciones.
- Threads.

2.6.4 Programando una JavaCard

Para poder programar una JavaCard es necesario contar con el kit de desarrollo provisto por Schlumberger el cual contiene un lector, un CD de instalación y una JavaCard. El CD de instalación contiene un software que permitirá realizar el programa que será cargado a la tarjeta JavaCard FrameWork (JCF) y otro que permitirá hacer la carga de dicho programa (MakeSolo).

Para comenzar a programar es necesario tener instalado JDK, ya que este ayudará a compilar el programa en conjunto con el JCF. El programa creado, sigue la misma estructura que un programa hecho en Java. Una vez compilado el programa (sin errores) es necesario ejecutar la sentencia mksolo, que generará un archivo en binario, el cual se carga a la JavaCard. A continuación se explican detalladamente estos pasos.

2.6.4.1 JavaCard FrameWork (JCF)

Esta es una API para desarrollar programas que se ubicarán dentro de las tarjetas inteligentes. Sigue una lógica de programación similar a la del lenguaje Java.

Contiene solamente dos paquetes, que son los siguientes:

- javacard.framework
- javacardx.framework

El primero de ellos contiene las variables estáticas definidas por la ISO - 7816. Las variables estáticas con el prefijo SW_, definen constantes del estado de respuesta (SW) establecidas por la ISO – 7816. Las variables estáticas con el prefijo OFFSET_, definen constantes para ser usadas como índices dentro del buffer de APDU para accesar a la información de encabezados establecidos por la ISO – 7816. Como algunas de las variables más usadas se tienen las siguientes:

- COMMAND_LENGTH.- Define la longitud de un comando ISO.
- DELETE_FILE.- Código de instrucción para borrar un archivo.
- OFFSET_INS.- Código de instrucción APDU.
- OFFSET_P1.- Código de primer parámetro APDU.
- SW_FILE_FULL.- SW para indicar que no hubo suficiente espacio de memoria en el archivo.
- SW_DATA_INVALID.- SW que indica que el dato fue inválido.
- SW_NO_ERROR.- SW que indica éxito en el comando.

El segundo paquete _GPOS contiene todas las llamadas al sistema operativo nativo accesible vía un programa de aplicación en Cyberflex. Los valores regresados de los métodos definidos en esta clase son declarados como constantes en la *interface* ST. Como algunos de los métodos más utilizados se tienen los siguientes:

- CreateFile (byte[]).- Crea un archivo con los datos contenidos en el arreglo.
- Directory (byte[], byte).- Despliega el encabezado del archivo.
- Execute (short, byte).- Indica al programa una ejecución de un archivo.
- GetMessage(byte[], byte, byte).- Recibe un comando válido en los parámetros de la terminal usando el protocolo T=0.
- ReadBinaryFile(short, byte, byte[]).- Lee datos del archivo seleccionado iniciando en el índice indicado.
- SelectFile(short).- Selecciona un directorio o archivo del directorio actual.
- SendMessage(byte[], byte).- Envía un mensaje a través de la línea de comunicación.
- SendStatus(byte).- Construye dos bytes de SW y los almacena en un SW1SW2 interno del estado de error.
- SendSW1SW2().- Envía el SW interno sobre la línea de comunicación.

Hay muchas otras variables y métodos descritos en este API que se pueden consultar para un mayor conocimiento de lo que se puede hacer con las tarjetas. En este trabajo sólo se mencionan los que serán utilizados en el siguiente capítulo cuando se vea una aplicación real. En el anexo 5 está el programa que demuestra el uso de los métodos y variables mencionados arriba con comentarios breves sobre las líneas de código.

Para probar la eficiencia de este programa se captura en un archivo llamado sample.java, de la misma manera en que se haría un programa en Java, tomando en cuenta que se debe tener instalado en la computadora donde se vaya a desarrollar el kit de desarrollo JCF, además del JDK.

Al haber concluido la captura de este programa, o de cualquier otro que se haga posteriormente, se compila, como se vió en el tema de Programación Básica. Solamente hay que agregar una opción a esta instrucción de compilación:

javac –g sample.java

Si no se tuvo ningún error de compilación se dispondrá a pasar al siguiente tema, donde se indicará cómo cargarlo a la tarjeta. En caso de que se hayan tenido errores de compilación se tiene que regresar al programa para corregirlo.

2.6.4.2 MakeSolo

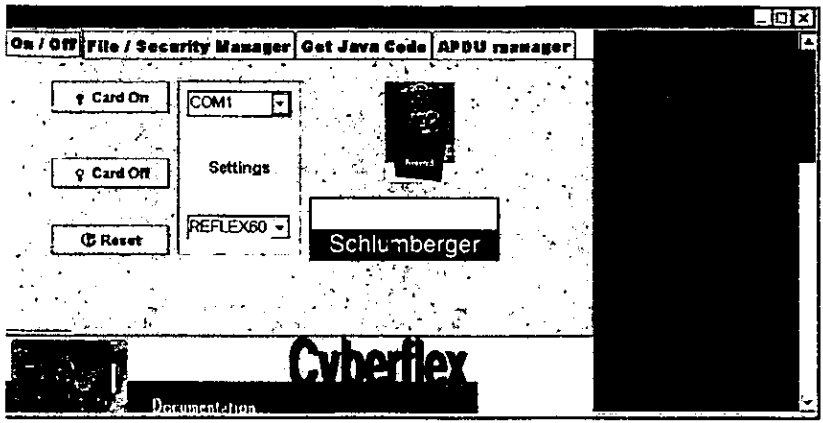
Al haber terminado de capturar el programa y haber tenido una compilación exitosa, lo que faltaría por hacer será únicamente ejecutar el comando mksolo, que es el que permitirá darle un formato al archivo compilado para poderlo cargar a la tarjeta. Para poder ejecutar este comando se debe asegurar que se tenga en la variable de ambiente PATH, además de que mksolo encuentre el archivo solo.upload, poniéndolo en el directorio donde se encuentra la aplicación o poner en la variable de ambiente UPLOADPATH, el directorio. Y se hará de la siguiente manera:

Desde la línea de comando hay que teclear:

mksolo –s sample.class

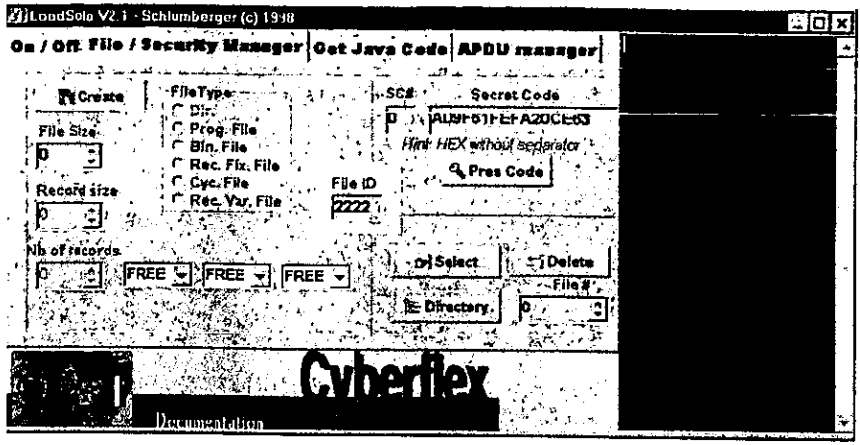
Esta instrucción generará un archivo .bin que es el que se cargará a la tarjeta mediante la aplicación de LoadSolo. LoadSolo consta de las siguientes partes como se muestra en la figura 2.14:

FIGURA 2.14 LoadSolo pestaña ON/OFF



En esta primer pantalla se tiene que presionar el botón "Card on", una vez que se haya conectado el lector de tarjeta e insertado la JavaCard en él. Se tiene la opción de conectarlo al puerto que se requiera, ya sea al com1, com2, etc..., además de seleccionar el lector que se tenga, que en este caso es el Reflex60 (para las tarjetas Multi8k). Una vez configurado el lector, se pasará a la siguiente pestaña la cual se muestra en la figura 2.15.

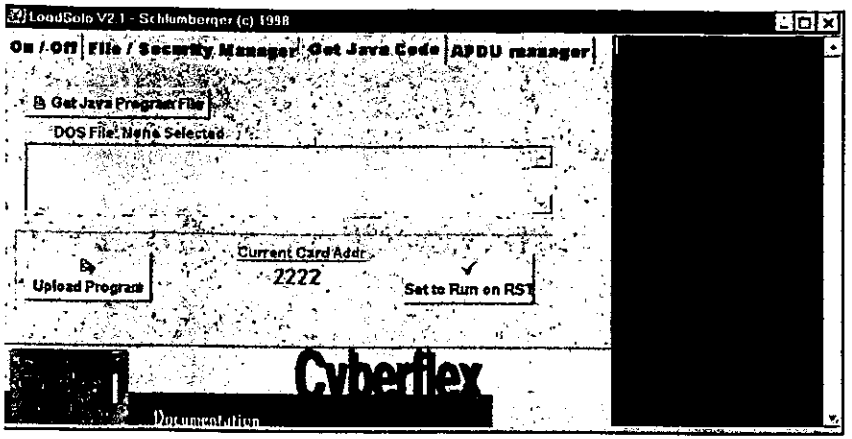
FIGURA 2.15 LoadSolo pestaña File/Security Manager



La tarjeta, por default tiene una configuración de estructura de archivos, por lo cual se puede navegar por sus directorios hasta la última rama, para que de ahí en adelante se pueda manipular como más convenga.

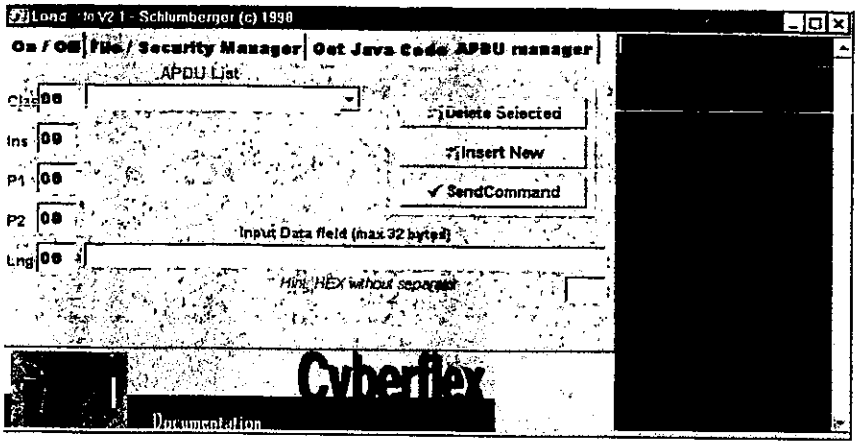
En esta pantalla se debe crear la estructura de archivos y un directorio donde se ubicará el archivo a importar, además de un nombre de archivo. Hay que seleccionar el tamaño, tipo y id, como opciones básicas. Cabe hacer notar que el tamaño de archivo que se introduzca, debe ser 16 bytes mayor al tamaño del archivo original y el tipo de archivo debe ser binario. El ID es un número cualquiera, tomando en cuenta que este número será el identificador con el cual se podrá saber con qué archivo se está trabajando. Después de haber hecho lo anterior y de haber seleccionado el archivo se pasará a la siguiente pestaña que contiene lo que muestra la figura 2.16.

FIGURA 2.16 LoadSolo pestaña Get Java Code



En este momento es donde se importará el archivo binario, creado con anterioridad al compilar el código fuente. Presionando el botón de Get Java Program File, se desplegará una ventana donde se podrá seleccionar el archivo. Después de esto, se oprimirá el botón de Upload Program para cargarlo a la tarjeta. Terminado esto, se le dará click al botón de Set to Run on RST, para iniciar el programa. Habiendo hecho esto se pasará a la siguiente pestaña que se muestra en la figura 2.17.

FIGURA 2.17 LoadSolo pestaña APDU manager



En esta pantalla se podrán dar las instrucciones que se quieran en el formato que en temas anteriores se vieron referente a APDU. Se puede ver la estructura de los comandos y armar instrucciones personalizadas. Esta pantalla es básicamente para Debug, es decir, para probar que la programación hecha en la tarjeta es correcta y así estar seguros de que hace lo que tiene que hacer.

Después de haber terminado con las pruebas de la tarjeta se pasará a crear la interfaz de usuario. Para realizar esto se tiene el software de Schlumberger llamado WinPractis, que es el que se utiliza para comunicarse con el lector de tarjeta. No sin antes haber hecho la pantalla utilizando lo ya aprendido de Java, como son los componentes gráficos Button, List, TextField, entre otros.

Y para comenzar a programar en WinPractis se dará un breve resumen a los principales métodos y variables de este software, en el siguiente tema.

2.6.5 Programación de la Comunicación con el Lector

WinPractis provee todas las funciones que son necesarias para comunicarse con los lectores de tarjetas. WinPractis consta de un archivo tipo DLL (Dynamic Library Link), que es el que contiene todos los métodos y variables utilizados para programar. El archivo a utilizar se llama scolv30x.dll. Cabe hacer notar que cómo el archivo a utilizar es una librería dinámica, los parámetros estarán dados en formato del lenguaje C, por tanto sería recomendable tener a la mano un manual de referencia del lenguaje C, para así saber de qué tamaño son los tipos de datos. También es importante saber que se tiene que encontrar un valor similar en Java para no tener errores de concordancia.

Las siguientes definiciones, abreviaciones y acrónimos serán de gran utilidad para poder dar inicio a la programación de la comunicación con el lector.-

Definiciones

IN Indica que el parámetro es un parámetro de entrada
OUT Indica que el parámetro es un parámetro de salida
0x00 Valor 0 en hexadecimal y de manera semejante el 1. . .

Abreviaturas

API Interfaz del programa de aplicación.
ATR Respuesta a incialización
DLL Librería de Liga Dinámica
PTS Tipo de Protocolo Seleccionado

2.6.5.1 Constantes y Valores de Retorno.

Variables de estado

SLBAPI_SYSTEMERROR.- Error de Sistema. Valor 0x00
SLBAPI_UNVAILABLE.- El lector o el puerto no están disponibles. Valor 0x01
SLBAPI_CARDABSENT.- El lector no contiene la tarjeta. Valor 0x02
SLBAPI_CARDSALLOWED.- La tarjeta está en el lector, lista para ser usada. Valor 0x03
SLBAPI_NEGOTIABLEMODE.- La tarjeta ha sido inicializada y espera el protocolo de comunicación. Valor 0x06
SLBAPI_PROTOCOLERROR.- Hubo un error con el protocolo T=1. Valor 0x10

Variables de ejecución: códigos de error

SLBAPI_OK.- El comando fue exitosamente ejecutado. Valor 0x0B
SLBAPI_ERROR.- Un error de sistema ha ocurrido. Valor 0x0A

Tipos de protocolo

SLBAPI_PROTOCOLT0.- La tarjeta trabaja con el protocolo T = 0. Valor 0xF0
SLBAPI_PROTOCOLUNDEFINED.- Protocolo Indefinido. Valor 0xFF

Tipos de Lector

SLBAPI_RT_SERIAL.- Lector serial tal como el Reflex60 o el SRC60. Valor 0x00

2.6.5.2 Funciones (Métodos).

int SLBAPIAllocate(IN const char *NombreLector,
IN const char *NombrePuerto,
OUT UINT *Lector,
OUT LPDWORD estadoLector)

Descripción: Habilita un manejador de lector y determina el estado del lector.

int SLBAPICancel(IN UINT Lector,
OUT LPDWORD estadoLector)

Descripción: Cancela la inserción o extracción de tarjeta. Envía un comando de abortar al lector y establece el estado del lector.

int SLBAPIPowerUP(IN UINT Lector,
OUT LPDWORD estadoLector)

Descripción: Aplica el voltaje a la tarjeta e inicializa la tarjeta, si no se le ha hecho antes.

int SLBAPIReset(IN UINT Lector,
OUT LPDWORD estadoLector,
OUT LPBYTE atr,
OUT LPDWORD longitudAtr)

Descripción: Se inicializa la tarjeta. El protocolo es dfinido.

int SLBAPISendIsoInT0(IN UINT Lector,
IN LPBYE Dato,
IN UINT TamañoDato,
OUT LPBYTE EstadoLector,
OUT LPBYTE StatusWord1,
OUT LPBYTE StatusWord2)

Descripción: Envía el comando a la tarjeta que trabaja con el protocolo T = 0 y regresa el status word.

int SLBAPISendIsoOutT0(IN UINT Lector,
IN LPBYTE Dato,
IN UINT TamañoDato,
OUT LPDWORD EstadoLector,
OUT LPBYTE Respuesta,
IN OUT UINT *TamañoRespuesta,
OUT LPBYTE StatusWord1,
OUT LPBYTE StatusWord2)

Descripción: Envía el comando a la tarjeta que trabaja con el protocolo T = 0 y lee la respuesta de la tarjeta y el Status Word.

BOOL SLBAPIGetReaderName(IN UINT NumeroLector,
OUT char far *NombreLector,
IN UINT Tamaño,
OUT BYTE *Tipo)

Descripción: Regresa el nombre del lector y su tipo.

La documentación completa respecto a estos métodos y variables se puede encontrar en el manual de referencia de este producto, que es otorgado en la compra del WinPractis 2.0 (o versiones más recientes).

El siguiente ejemplo sirve para demostrar cómo se usan estos métodos:

Ejemplo de cómo mandar instrucciones a la tarjeta

Por ser WinPractis un archivo con extensión DLL, es factible usarlo con cualquier sistema integrado de desarrollo para Windows, por tal motivo el ejemplo que se muestra está creado en VisualBasic, ya que así podemos tener una mejor visión de cómo trabajan estos métodos. En el siguiente capítulo se verá cómo es utilizado WinPractis con el lenguaje Java.

```
Function SeleccionaPF(ByVal Address As String, ByRef Answer As String) As Boolean
    //Declaración de variables locales
Dim ret     As Integer
Dim Temp    As String
Dim ISO     As String
Dim SW1     As String
Dim SW2     As String
Dim I       As Integer
Dim Size    As Integer
Dim Regreso As Boolean

Temp = "00A4000002" & Address   //contiene el comando APDU
ISO = ""
sSW1 = " "
sSW2 = " "
```

```
//Convierte a Binario
  For I = 1 To Len(Temp) Step 2
    ISO = ISO + Chr(Val("&h" + Mid$(Temp, I, 2)))
  Next I
  Size = Len(ISO)
  //llamada al metodo que envia la instrucción al lector
  ret = SLBAPISendIsoInT0(hReader, ISO, Size, IState, sSW1, sSW2)

//convierte respuesta a ASCII
  If (ret = SLBAPI_OK) Then
    Temp = Hex$(Asc(Mid$(sSW2, 1, 1)))
    If Len(Temp) = 1 Then Temp = "0" + Temp
    Answer = Hex$(Asc(Mid$(sSW1, 1, 1))) + Temp

//Se arma la variable de retorno, si es que no hubo error
    If (Answer = "9000") Then
      SeleccionaPF = True
      Exit Function
    Else
      SeleccionaPF = False
      Exit Function
    End If
  Else
    Answer = ""
    SeleccionaPF = False
  End If

End Function
```

CAPÍTULO III

AUTENTIFICACIÓN DE USUARIOS MEDIANTE TARJETAS INTELIGENTES

3.1 ANTECEDENTES

Las compañías de seguros han implementado infinidad de recursos y medios para dar una forma de identificación a sus asegurados, y de esta manera protegerse de fraudes y amparar a sus asegurados al momento de tener que hacer válidas sus coberturas.

Desde la mitad del siglo pasado, las primeras compañías de seguros implementaron la "póliza", documento en el cual se estipulaban las reglas del contrato que celebraban la compañía de seguros y el asegurador. En estas reglas se explicaba en qué condiciones la compañía de seguros estaba responsabilizada en cubrir el monto de la póliza que había expedido a su asegurado. En esta póliza también se describía qué tipo de seguro se tenía contratado, el monto y la fecha de vigencia de la misma. Este tipo de póliza se implementó para el seguro naval.

Como era de esperarse, el documento no tardó en ser falsificado por defraudadores de seguros, por lo cual se empezó a implementar el registro de seguros. Este registro se hacía en una regidora de seguros la cual anotaba el número de la póliza en una central y a la póliza expedida al asegurado se le imprimía un sello real.

Esta práctica cada vez fue más difícil ya que las compañías de seguros empezaron a introducir al mercado diferentes tipos de seguros, lo cual dificultaba llevar un control estricto para evitar el fraude en las pólizas.

El problema a través del tiempo ha tratado de ser atacado de diferentes formas, sustentado por documentos oficiales que no siempre son de buena procedencia, es decir, las compañías aseguradoras tienen que estar luchando contra bandas de defraudadores, quienes sin escrúpulo alguno y coludidos muchas veces con personal interno de las compañías de seguros, así como con personal gubernamental, expiden documentos oficiales que perjudican a las aseguradoras.

3.2 Problemática Planteada

Las compañías de seguros han implementado numerosos y costosos planes para impedir ser sujetos de fraudes. Las aseguradoras llegan a tener más de un millón de clientes que se atacan de diferentes formas desde un punto de vista comercial, es decir, una persona puede tener diferentes paquetes de seguros contratados con una misma aseguradora. Esto ocasiona que en algunos casos la aseguradora no tenga forma de identificar a la persona para ligarla con otros planes de seguros ya contratados por la misma.

Aunque las aseguradoras han invertido fuertemente en costosos equipos de cómputo y software, se encuentra que el negocio de los seguros avanza mucho más rápido de lo que ellos implementan la tecnología.

Esto ocasiona que no siempre se tengan bases de datos actualizadas y completas. La redundancia en la base de datos es un problema típico en una compañía de seguros.

En el caso de un seguro de vida, se tienen diferentes tipos de problemas para identificar a las personas que tienen derecho al servicio que ofrece el seguro.

El servicio del seguro médico ofrece la consulta de un médico general que se encuentre dentro de un directorio que proporciona la aseguradora al asegurado al momento de afiliarse.

Este médico familiar tiene la obligación de hacer un reporte mediante un formato que le proporciona la aseguradora para hacer un historial del paciente y así saber qué padecimientos médicos ha tenido el paciente y saber si la enfermedad entra en la póliza que contrató el paciente o no.

Estos tipos de reportes se envían por paquetería a la central de la aseguradora por medio de valija, es decir, un conjunto de formas con los datos de diferentes pacientes. Cuando llegan a la central de seguros estas valijas son capturadas por un área especializada en este concepto y se actualizan las bases de datos. Cabe señalar que existen innumerables errores en la captura de datos, desde el médico que ingresa la información hasta los capturistas de la misma aseguradora.

Estos errores no son la única problemática que se presenta en este tipo de seguros ya que en algunas ocasiones se pierden valijas completas de formas enviadas por médicos generales.

En ocasiones en donde el paciente se siente muy mal de salud o el médico general le indica que es necesario trasladarlo a un hospital empieza otro tipo de problema ya que muchas veces el asegurado no lleva los papeles de su póliza en donde se encuentra la cobertura para gastos médicos mayores. Cuando sucede este tipo de cosas el paciente se puede encontrar en diferentes situaciones: 1) que no reciba la atención por parte del hospital y se tenga que trasladar a otro con todos los problemas que esto atrae; 2) que el hospital lo atienda y cuando revise la cobertura de la póliza del asegurado descubra que no se cubre por la aseguradora el tratamiento o la atención que se le esté brindando y pida una garantía para el pago de sus servicios; 3) que el paciente tenga que cubrir sus gastos y después con todas

las facturas que dé el hospital, empezar con difíciles trámites para que la aseguradora reembolse la cantidad entregada por el paciente al hospital, etc.

En estas situaciones los problemas son principalmente para los asegurados. Cuando los problemas son para las aseguradoras existen tremendos fraudes, ya que en ocasiones personas que cuentan con un servicio médico de este tipo se dedican a dar el servicio a otra gente que no lo tiene contratado, por ejemplo un empleado de una institución financiera tenía registrado en su paquete de gastos médicos familiares a cuatro hijos a los cuales llevaba con un médico familiar a un consultorio, pero se le ocurrió que por un poco más de lo que él le pagaba al médico como deducible podía llevar a otros niños que no tenían este servicio, es decir, él cobraba por llevar al médico a niños que no eran suyos. El problema es que las aseguradoras no expiden credenciales por persona a este tipo de personal financiero, ya que son de confianza, por lo tanto el servicio médico se les proporciona con solo presentar su póliza y su credencial de la institución financiera en donde labora. Este tipo de fraude es muy difícil de detectar, por lo que a las aseguradoras les cuesta miles de millones de pesos al año y sigue en aumento.

En el país se han detectado más de ciento cincuenta de estos casos pero se piensa que existen miles que no se pueden detectar.

Otro fraude muy común dentro de las aseguradoras es el pago de medicamentos, ya que no todas las aseguradoras han implementado una forma efectiva para llevar control en el surtido de recetas médicas, en muchos de los casos la misma receta se ha llegado a surtir hasta doscientas veces en diferentes farmacias, en diferentes días. Este tipo de fraude se ha detectado a través del número de folio de la receta, la fecha y el nombre del médico. Este fraude es muy común pero muy difícil de detectar por el tipo de control que se lleva actualmente.

La parte más difícil de solucionar es encontrar el medio de almacenamiento de información más adecuado, ya que en nuestro país hay bandas organizadas que se dedican a falsificar cualquier tipo de documentos oficiales con la más alta calidad.

Estas bandas no solo falsifican documentos con cualquier tipo de seguridad como papel con gota de agua o transparencias, sino también han incursionado con tarjetas magnéticas como las tarjetas de crédito, lo que hace más difícil evitar la falsificación.

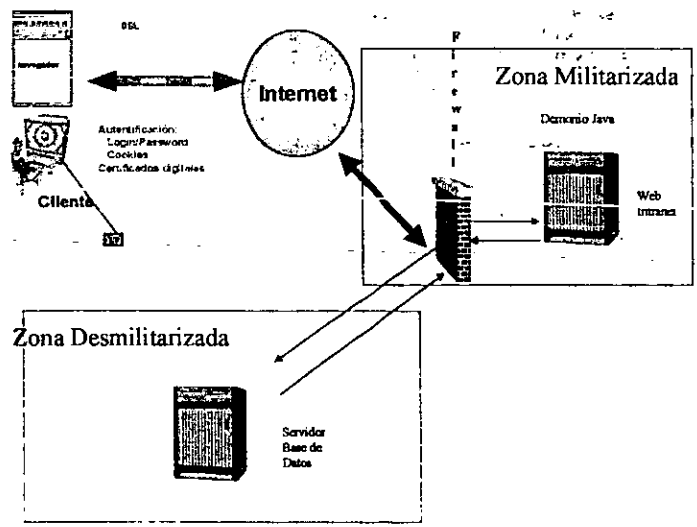
Sin embargo el problema mayor es tener la información en tiempo real o por lo menos que la información siempre esté actualizada.

La inversión que tienen que hacer las aseguradoras para este tipo de servicio (información en tiempo real) con una arquitectura tradicional y una línea telefónica dedicada es muy costosa, ya que no se tendría el mismo número de transacciones que las bancarias, pensando en el caso en que una aseguradora llegara a tener más de diez mil médicos afiliados a sus servicios a escala nacional y faltaran por integrar hospitales y farmacias que estuvieran dentro del paquete médico. Este tipo de "*arquitectura tradicional*" es el siguiente: en una terminal cliente se pone una lectora de tarjeta de banda magnética, como las utilizadas en almacenes de autoservicio denominadas terminales *Visa*. Estas terminales están conectadas a un servidor de transacciones vía red telefónica pública, en este servidor se atienden las peticiones de acuerdo al tipo de tarjeta que se está deslizando por la lectora de la cinta magnética. La terminal sabe quién es el banco expedidor de la tarjeta por medio de un algoritmo que obtiene la clave del banco de acuerdo a los primeros 8 dígitos de la tarjeta. Este servidor a su vez se comunica al servidor del banco en cuestión para solicitar la autorización del cargo y regresar la petición hasta el punto de venta.

3.3 Solución Propuesta

Para la problemática descrita anteriormente se propone utilizar tarjetas inteligentes bajo un esquema de Internet - Intranet con lectoras de tarjetas inteligentes en las terminales de servicio. Este tipo de solución se describe a continuación en la figura 3.1.

Figura 3.1 Arquitectura de Solución



Con esta solución se plantea utilizar tecnología de punta en la cual los costos se reducirían hasta un 65%, ya que los establecimientos afiliados cuentan ya con la mayoría del equipo necesario.

Para llevar acabo esta solución es necesario que las clínicas, hospitales y médicos afiliados tengan una computadora personal con por lo menos Windows95, cualquier Visualizador de web (netscape o explorer) y una lectora de tarjeta (Reflex60 de Schulmberger), la cual será independiente y se conectará al puerto serial de la máquina o una lectora integrada al teclado de la máquina.

En las oficinas de la compañía aseguradora se necesita una arquitectura de Internet - Intranet como se demuestra en el diagrama anterior.

En el diagrama anterior se muestra la infraestructura con la que cuentan actualmente casi todas las aseguradoras de este país.

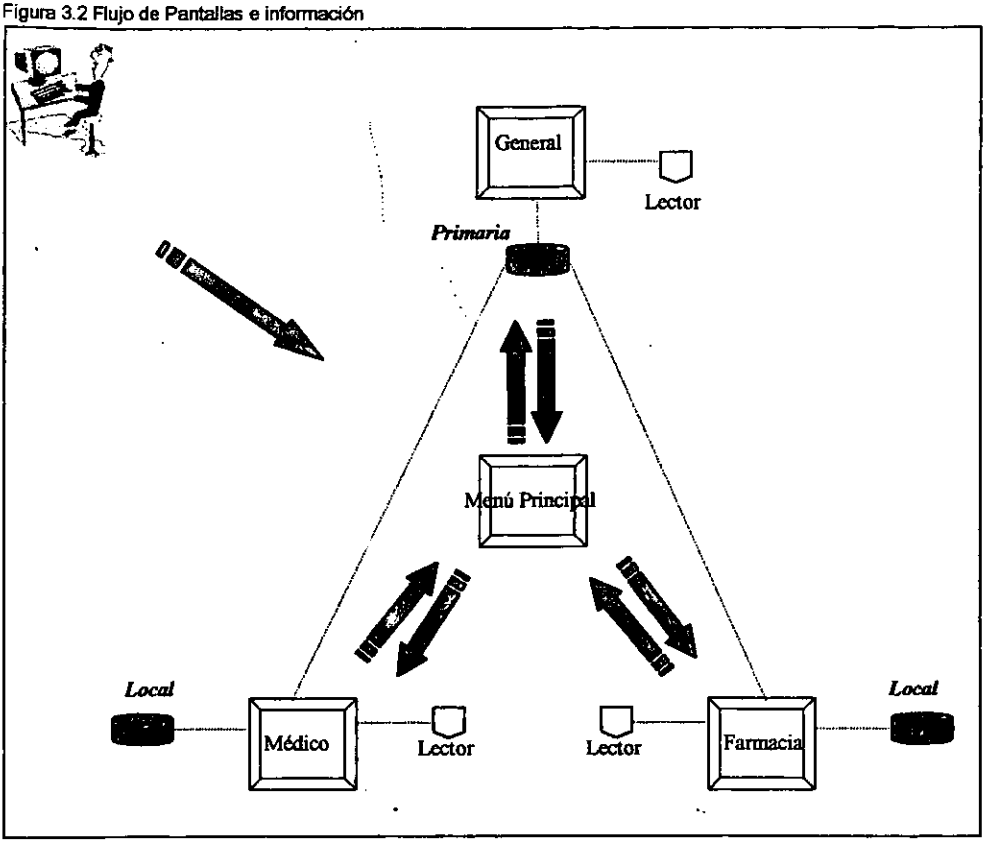
La compañía aseguradora proporciona a los usuarios de su sistema un Login y password para su Intranet que podrán acceder desde una dirección de Internet; el sistema viajará vía Internet - Intranet hasta la máquina de sus clientes. El sistema deberá estar desarrollado en lenguaje Java, el cual nos permitirá tener comunicaciones seguras y rápidas.

3.4 Detalle de la aplicación

El modo de operación del sistema será el siguiente: los usuarios se conectarán a la aplicación cada vez que llegue algún asegurado con su tarjeta a solicitar el servicio. En la tarjeta inteligente vienen los datos generales de cada persona como son: nombre completo, suma asegurada, deducible, cobertura, fecha de última visita al médico, tipo de sangre, fecha de antigüedad, fecha de vencimiento de la póliza y fotografía, algunos de los cuales son solo almacenados y no son visibles en la aplicación.

Antes de explicar a detalle el funcionamiento de la aplicación convendría primero ver a grandes rasgos y por medio de un diagrama de flujo de información, cómo sería el funcionamiento de la aplicación. Se tienen cuatro pantallas: la pantalla que contiene al menú principal, desde donde se accesa a otra pantalla específica dependiendo del tipo de usuario que ingrese al sistema; la pantalla para el personal farmacéutico, en la cual se podrá surtir la medicina recetada por el médico; la pantalla para los médicos, quienes podrán ver ahí la fecha de última visita, diagnósticos anteriores, además de que se le darán las especificaciones al paciente respecto a su consulta; y por último la pantalla para el personal de la aseguradora, quien podrá modificar los datos generales de la persona afiliada, si así lo desea. Enseguida se verá, en el diagrama 3.2, cómo fluye dicha información y cuál es el flujo de pantallas.

Figura 3.2 Flujo de Pantallas e información

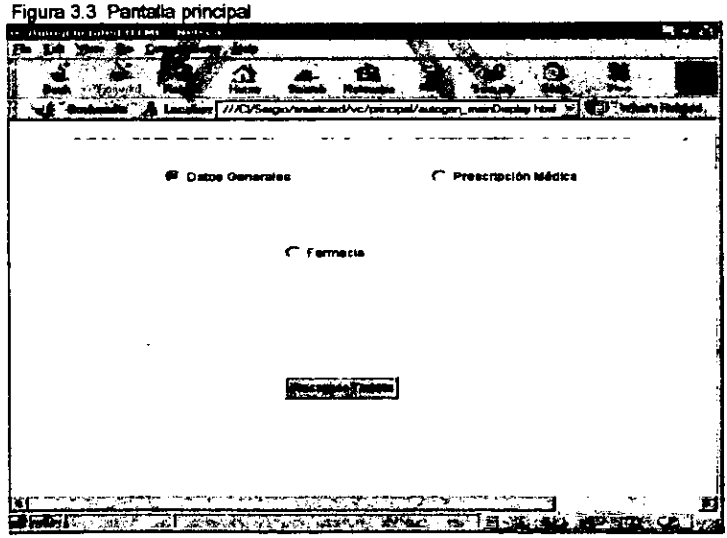


Las flechas verdes indican el flujo de las pantallas, es decir, la manera en que se puede "navegar" dentro del sistema. Las líneas punteadas de color rojo indican cuando se hace una operación en la Base de Datos, ya sea en la local o en la primaria que es de donde se obtiene la información general del asegurado. Se puede notar entonces, que la aplicación está comunicándose a dos Bases de Datos simultáneamente sin tener que desconectarse de una para atender a la otra. Y por último las líneas punteadas de color azul indican cuando se hace una operación en la Tarjeta Inteligente.

La aplicación inicia cuando el usuario abre su navegador (Netscape, IExplorer, etc...) en la página donde se encuentra dicho sistema. Como se puede ver en la gráfica anterior, la primer página mostrada es la del Menú Principal, que es la que tiene las opciones para poder navegar de pantalla en pantalla. Entonces se contemplan tres tipos de personal que utiliza el sistema: el médico, el personal farmacéutico y el personal de la aseguradora que se encargaría de modificar los datos generales del asegurado. Si la persona que accesa al sistema es, por ejemplo, un Médico entonces sólo podrá hacerlo en la pantalla de Consultas Médicas, pasando primeramente por el menú principal. La descripción detallada de las pantallas se hace a continuación:

Cuando se entra a la aplicación, la primer pantalla que se ve, es la mostrada en la figura 3.3.

Figura 3.3 Pantalla principal

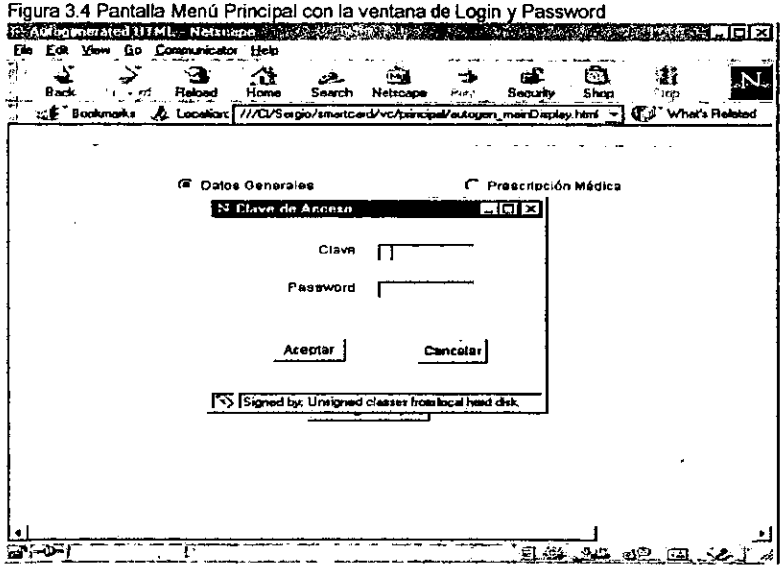


Lo primero que se tiene que hacer para entrar en cualquiera de las opciones mostradas es, seleccionar el botón de "Recargar Tarjeta", ya que es el que iniciará la aplicación con los datos contenidos en la tarjeta[1].

Después se podrá seleccionar la opción a la que se esté permitido, dependiendo de la Clave o Login asignada. Estos son independientes de la tarjeta ya que son asignados por la aseguradora (dependiendo el título del empleo).

Si el usuario en cuestión fuera personal directo de la aseguradora podría sólo accesar a la pantalla de Datos Generales, no sin antes haber introducido su Login y Password correspondiente (figura 3.4).

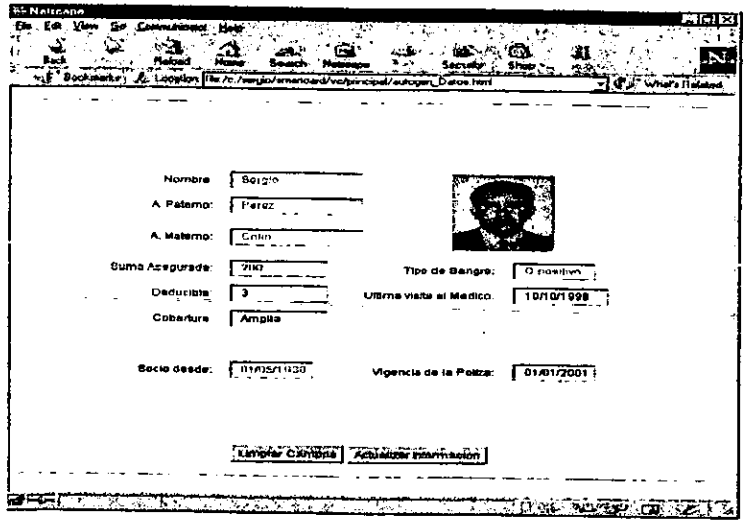
Figura 3.4 Pantalla Menú Principal con la ventana de Login y Password



[1] En el anexo 2 se tiene el código de esta pantalla que muestra la conexión del applet hacia la tarjeta y viceversa.

Las validaciones de Login y Password se hacen enviando la información a través de Internet, pasando por el firewall, que verifica si ese diente tiene acceso a la intranet, hasta llegar al servidor de Web. Este servidor tiene una aplicación hecha en Java que resuelve las requisiciones hechas por los clientes. La aplicación Java hace una conexión hacia el servidor de Base de Datos que es donde se compara el Login y Password ingresados para ver si existen y son ambos correctos (figura 3.2). La pantalla de Datos Generales se muestra a continuación (figura 3.5).

Figura 3.5 Pantalla Datos Generales

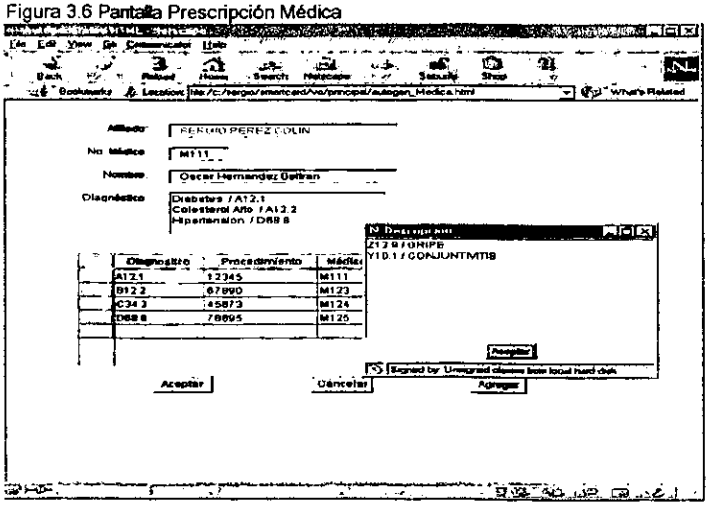


En esta pantalla se tienen los datos generales del afiliado. Estos datos son obtenidos de la tarjeta por medio de claves asociadas a descripciones, por ejemplo el campo Tipo de Sangre está en la tarjeta como 1 y cuando se carga la información se busca el significado de esa clave en la aplicación. Aquí el personal de la aseguradora tendrá la opción de modificar algunos campos para tener actualizada tanto la tarjeta del afiliado como sus datos generales y personales. Si fuera un super usuario tendría la opción de modificar todos los campos deseados. Habiendo actualizado la información regresará al menú principal donde podrá actualizar a otro afiliado.

Cuando se hace la operación de actualización, los datos que están visibles se cambian a sus claves para así modificar el contenido de la tarjeta y la base de datos. Es decir, el campo Tipo de Sangre no se puede almacenar como "0 Positivo" y en lugar de eso se pone la clave relacionada que es "1". A esto se le conoce como compactación de datos y sirve para disminuir el tamaño físico a ocupar dentro de la tarjeta. El código fuente de esta pantalla se puede ver en el anexo 3, que se encuentra al final de este trabajo.

Ahora bien, desde el menú principal, si el usuario fuera un médico, entonces tendría acceso solo a la opción de Prescripción Médica, cuya pantalla se muestra a continuación (figura 3.6).

Figura 3.6 Pantalla Prescripción Médica



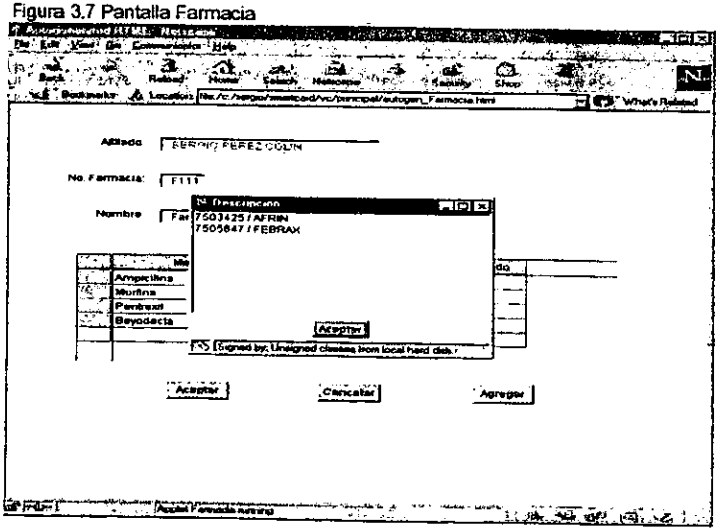
En esta pantalla, el médico puede visualizar los diagnósticos pronosticados anteriormente junto con su procedimiento correspondiente. Podrá agregar además una lista de diagnóstico – procedimiento correspondiente al afiliado en cuestión.

Todo diagnóstico tiene un procedimiento asociado con una clave. Todas estas claves son traídas de la Base de Datos la primera vez que se carga la aplicación.

En la rejilla también se puede ver el estado del cumplimiento de la relación diagnóstico – procedimiento. Habiendo terminado la consulta el médico hará click en el botón de "Aceptar" para actualizarle la tarjeta con una clave de la consulta, además esta clave será actualizada en la Base de Datos junto con los diagnósticos para que cuando vaya a la farmacia, esta pueda tener esa relación. Después de haberse actualizado la tarjeta se imprimirá la receta respectiva que se le entregará al afiliado, además de que la aplicación regresará a la pantalla principal de menú.

Ya estando en el menú principal, si el usuario fuera un farmacéutico, entonces podría solo entrar a la opción de "Farmacia". Esta pantalla se muestra a continuación en la figura 3.7.

Figura 3.7 Pantalla Farmacia



En esta pantalla el personal de la farmacia podrá ver las medicinas surtidas con anterioridad, teniendo visible en la rejilla, además de las medicinas surtidas, la clave respectiva y el estado de atención.

Las medicinas que serán surtidas aparecerán en la rejilla, además de que se podrán dar de alta otras medicinas con el botón de agregar, y lo que va a suceder es que a la rejilla se le incrementará un renglón para que se pueda introducir la medicina a surtir. Las medicinas recetadas son cargadas a la aplicación desde la Base de Datos dependiendo de la clave que le fue asignada al afiliado cuando estuvo en consulta. Por razones internas de las farmacias el afiliado tendrá que llevar la receta expedida en el consultorio médico para que el farmacéutico pueda agregarla a su Base de Datos local con efectos de comprobación fiscal y con efectos de comprobar la medicina a surtir. Habiendo concluido la atención al cliente y presionando el botón de "Aceptar", se regresará a la pantalla de menú principal donde podrá atender a otro cliente.

Como se puede ver, esta aplicación hace muy simple el manejo de seguridad y consistencia de los datos de la aseguradora manipulándolos de una manera eficaz, resolviendo así los problemas acaecidos por falta de una tecnología capaz de respaldar sus operaciones.

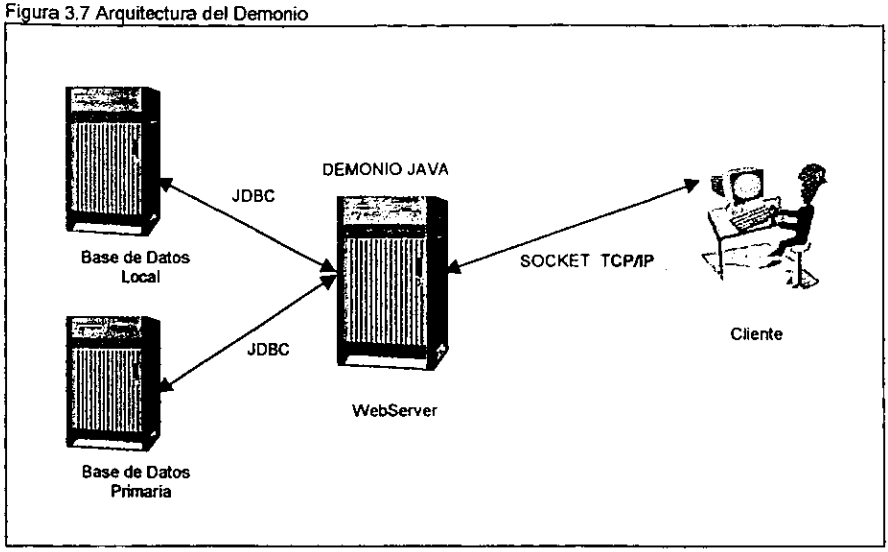
Como se mencionó al principio de este trabajo, específicamente en el Objetivo, la finalidad no es la de analizar a detalle este sistema sino más bien la de ejemplificar cómo con la tecnología de las tarjetas inteligentes trabajando en conjunto con el lenguaje Java puede tener un sistema confiable en cuestiones de seguridad, eficaz en cuanto al tiempo de ejecución y seguro en lo que respecta a la consistencia de los datos.

Pero es importante aclarar que no se toma esto como excusa para no mostrar por completo dicho sistema, (cuando se dice "completo" se trata de decir mostrar el sistema desde la parte del modelo y diseño de la Base de Datos, configuración del WebServer, análisis y diseño de la aplicación, hasta llegar al desarrollo pantalla a pantalla de la aplicación), de tal forma que se explicarán algunos de los puntos no vistos, en forma breve, en los siguientes párrafos.

Primeramente se verá la parte de la Base de Datos. La Base de Datos está creada en Oracle 7.8. Existen tres Bases de Datos diferentes para cada una de las pantallas en cuestión y una más para la aseguradora, es decir, cuando el usuario se encuentra en la pantalla de Farmacia el programa accesa a esa Base de Datos para obtener información acerca de la medicina a surtir además de que tiene que hacer un acceso a la Base de Datos primaria que se encuentra en la aseguradora, de donde se obtiene la información general del cliente. Lo mismo sucede en la pantalla del Medico y en la de Datos Generales.

Por otro lado, en el WebServer hay una aplicación hecha 100% en Java que es la que tiene una comunicación abierta con la Base de Datos, es decir, esta aplicación tiene un Socket siempre abierto hacia el lado de la Base de Datos y otro que se abre hacia el lado del cliente siempre y cuando algún cliente se conecte con el WebServer. Este programa hecho en Java es un programa que siempre se está ejecutando esperando las peticiones de los clientes. A este tipo de programas se le conoce como "demonios". Esto se puede apreciar mejor en la figura 3.7 donde se muestra gráficamente la comunicación del "demonio" hecho en Java, tanto con el cliente como con la Base de Datos.

Figura 3.7 Arquitectura del Demonio



Todas las peticiones que el cliente haga hacia el WebServer, las va a estar atendiéndo es el demonio Java. Cabe recordar que este demonio solo devuelve datos específicos de la Base de datos (descripción de alguna clave), ya que la parte principal está almacenada en la Tarjeta Inteligente. Como parte principal se debe entender los datos generales del asegurado - nombre completo, suma asegurada, deducible, cobertura, fecha de última visita al médico, tipo de sangre, fecha de antigüedad, fecha de vencimiento de la póliza y fotografía -. En el anexo 4 se tiene un esquema de la estructura de directorios que se crearon dentro de la tarjeta, así como el tamaño de los mismos.

CONCLUSIONES

Al hablar de fraude en este trabajo no se trató de dar a conocer algo nuevo en este país. Lamentablemente el fraude y la corrupción existen en un gran número de instituciones gubernamentales y privadas, pero lo peor de todo es que nos quejamos de ello cuando los que la propiciamos somos nosotros. Imaginemos al conductor que va por la calle en su vehículo y éste se pasa un alto y lo detiene la policía. Aproximadamente el 90% de las personas, puestas en su lugar, optarían por sobornar al oficial antes de que les levanten una infracción para no tener que ir a perder tiempo a las oficinas a pagarla. Imaginemos también a la persona que va a sacar su licencia de conducir. Esta persona prefiere pagar una cantidad extra de dinero para tramitar su licencia que esperar a aprobar los exámenes que le son aplicados.

Cuanto más avanza la tecnología, más avanza el número de personas que se dedica al fraude y las formas de hacerlo. Cuando se inventaron las tarjetas de crédito se inventaron formas de falsificarlas. El crimen organizado logró su cometido en los primeros tiempos de vida de las tarjetas pero estas tuvieron que desarrollar un sistema anti – falsificación. Si se edita un nuevo documento se inventa una nueva forma de falsificarlo. ¿Hasta dónde iremos a parar?, ¿Por qué no creamos conciencia de que eso es degradante para una persona?, ¿Queremos, así, crecer como nación?, ¿Queremos, así, que nuestros gobernantes no se enriquezcan a costa de nosotros?, ¿Cómo queremos eliminar la corrupción?. Creo que es necesario cambiar nuestra forma de vida en el sentido amplio de cómo nos comportamos con los demás y cómo queremos que nos traten.

A pesar de que este trabajo no fue diseñado para analizar la parte psicológica del hombre, cabe mencionar que es importante que la gente cree conciencia de la forma de vida que lleva y de cuánto está dispuesto a dar por los demás y por su país.

Cuando se habló de una tarjeta inteligente para solucionar el problema de fraude en las aseguradoras se trató de hacer ver, de alguna forma, que se puede diseñar un tipo de arquitectura diferente al que planteaba la aseguradora para resolver el problema. Por otro lado, también se pensó en la posibilidad de que el cliente invirtiera en tecnología y recursos humanos, porque muchos de los clientes pretenden obtener una solución con los recursos que tienen actualmente y no se dan cuenta de que pueden estar gastando más dándoles soporte a equipos viejos que actualizarlos.

Como un ejemplo claro se tiene el lenguaje Java, en donde las nuevas versiones de éste solo corren con determinadas versiones de Browser's, y no es posible brindarles una solución rápida y sencilla puesto que hay que seguir desarrollando aplicaciones con versiones "viejas" de este lenguaje, ya que se tienen máquinas obsoletas.

Actualmente existe la versión del JDK 1.2.2 que ya incluye nuevos componentes, modificaciones a las clases ya existentes y que en algunos casos ya no son compatibles con versiones anteriores.

Cuando se elaboró esta Tesina, se tomó como base el JDK versión 1.0.2 para los conceptos y documentación, pero para el desarrollo se utilizó también la versión de JDK 1.1.x.

Como podemos ver las versiones de JDK han avanzado muy rápido, con relación a los Browser's que lo pueden soportar. Esto es un problema cuando se quiere desarrollar en la versión más actual, ya que es casi seguro que los Browser's no soportarán sus nuevos componentes.

Más allá de los problemas que nos podamos encontrar con el uso de nuevas versiones, se puede uno dar cuenta qué tan rápido avanza este lenguaje y más aún, cómo se va haciendo más seguro y confiable. Java al mismo tiempo nos facilita caminos, no sólo, para detectar fraudes, sino para evitarlos por medio de las tarjetas inteligentes. Sin embargo, Java tiene una inmensidad de aplicaciones como: Banca Virtual, Comercio Electrónico, Educacional, etc.

Es por eso que recomiendo el uso y enseñanza de Java en las escuelas donde se tenga la idea de implantar un lenguaje fácil de aprender, que sea para usos múltiples, robusto y seguro. Y qué mejor que Java para hacer eso. Puesto que Java es Orientado a Objetos, ligero y multiplataforma tenemos la seguridad de que estamos aprendiendo un buen lenguaje.

BIBLIOGRAFÍA

1. Heller, Philip y Simon, Roberts
   Java 1.1 Certification Study Guide
   Editorial Sybex. E.U. 1998

2. Lemay, Laura y L. Perkins, Charles
   Aprendiendo Java en 21 días
   Edición en español. Editorial Prentice – Hall Hispanoamericana. S.A. México 1996

3. Rithey, Tim
   Programming with java!
   Edición en inglés. Editorial New Riders. E.U. 1998

4. Developers Series
   Cyberflex, the card for java, Programmers Guide
   Editorial Schlumberger. Francia 1998

5. Sun Educational Service
   Java programming (SL-275)
   Editorial Sun Microsystems. E.U. 1996

CONSULTA INTERNET

6. http://www.redbooks.ibm.com/

7. http://www.chipcard.ibm.com/

8. http://www.idg.net/

9. http://www.opencard.org/

10. http://www.javaworld.com/

11. http://www.javasoft.com/

12. http://sun.java.com/

ANEXOS

ANEXO 1

Programas fuente de la tarjeta.

```
import javacard.framework.*; //importación del paquete de clases
import javacardx.framework.*;

public class Cardlet1 extends _GPOS implements ST,ISO
{
  public static void main(String arg[]) //método principal para ejecutar la
  {                                     //clase dentro de la tarjeta
    byte tbuffer[] = new byte[16];     //buffer donde se almacena la información a devolver
    byte rbuffer[] = new byte[502];    //buffer en donde se almacena la instrucción deseada

    Loyalty myLoyal = new Loyalty();   //Clase que inserta y obtiene
                                       //los datos del archivo

    Execute((short)0,(byte)0);         //manda el control al lector

    while (true)
    {
      GetMessage(rbuffer,(byte)5,(byte)0);  //Espera hasta que llegue una
                                            //instrucción
      if(rbuffer[1] == (byte) 0xF4)
      {
                Execute((short)0,(byte)0); //Regresa al cardlet de default

                tbuffer[0] = (byte) 0x90;
                tbuffer[1] = (byte) 0x00;
                SendMessage(tbuffer,(byte) 2); //envia mensaje de
      }                                        //respuesta al
      else                                     //Lector
      if(rbuffer[1] == (byte) 0xA4)
      {
                byte estado = myLoyal.add(rbuffer); //agrega los datos al
                                                    //archivo de la tarjeta
                      if(estado == ST.SUCCESS)
                      {
                     tbuffer[0] = (byte) 0x90;
                  tbuffer[1] = (byte) 0x00;
                     SendMessage(tbuffer,(byte) 2);
                      }
                      else
                      {
                  tbuffer[0] = estado;
                           SendStatus(ST.WRITE_ERROR); //Hubo un error
                      }
      }
      else
```

```
    if(rbuffer[1] == (byte) 0xA5)
    {
            tbuffer[0] = rbuffer[1];
            SendMessage(tbuffer,(byte)1); //se indica que va a devolver
                                                    //algún valor
            byte[] rBuffer = myLoyal.returnValue();//obtiene la informacion del archivo
                    if(rBuffer.length != 1)
                    {
                            SendMessage(rBuffer,(byte)(rBuffer.length));//Se envía la información
              tbuffer[0] = (byte) 0x90;
              tbuffer[1] = (byte) 0x00;
              SendMessage(tbuffer,(byte) 2); //devuelve el mensaje
                                    //de respuesta
              SendStatus(ST.SUCCESS);//envía mensaje de status
                    }
                    else
            {
                            SendStatus(ST.UNKNOWN_ERROR); //Hubo un error
            }
    }
    else
    if(rbuffer[1] == (byte) 0xA6)
    {
            myLoyal.reset(); //Limpia el contenido del archivo
            tbuffer[0] = (byte) 0x90;
            tbuffer[1] = (byte) 0x00;
            SendMessage(tbuffer,(byte) 2);
    }
    else
    {
            tbuffer[0] = (byte) 0x6D;
            tbuffer[1] = (byte) 0x00;
            SendMessage(tbuffer,(byte) 2);
    }

   } //while
  }//main
}//class



import javacard.framework.*;
import javacardx.framework.*;

public class Loyalty extends _GPOS implements ST,ISO
{
  byte  tokens[];

  public Loyalty()
  {
    SelectFile((short) 0x2223); //id del archivo
    tokens = new byte[1];
  }
```

```
public void reset()
{
  tokens[0] = (byte)0;
  WriteBinaryFile((short) 0,(byte) 1,tokens);//inicializa el archivo
}

public byte add(byte[] tk )
{
        byte status = (byte)0;
        status = WriteBinaryFile((short) 0,(byte) 1,tokens);//inicializa el archivo

        if(status != ST.SUCCESS)
                return status;

    status = WriteBinaryFile((short) 2,(byte) (tk.length), tk); //Escribe
                                                    //el arreglo de bytes al archivo

        return status;
}

public byte[] returnValue()
{
        byte status = (byte)0;
        byte[] rBuffer = new byte[4406];
        status = ReadBinaryFile((short) 0, (byte)113, rBuffer);//Lee el contenido
                                                              //del archivo
        if(status != ST.SUCCESS) //Si hubo algún error
        {
          rBuffer = new byte[1];
          rBuffer[0] = status;
          return rBuffer;
        }

        SelectFile((short) 0x2222); //id del archivo imagen
        status = ReadBinaryFile((short) 114, (byte)4406, rBuffer);
        if(status != ST.SUCCESS)
        {
          rBuffer = new byte[1];
          rBuffer[0] = status;
          return rBuffer;
        }

        return rBuffer;
  }
}
```

ANEXO 2

Programa de conectividad del Applet hacia la tarjeta.

```
import java.awt.*;          //paquete de clases a importar
import java.applet.*;

 public class CardletPC
{
  byte abyDatos[] = new byte[4406];

      //método que verifica la conectividad del lector
  public boolean AsignaLector()
  {
    int    RetValue = 0, hReader = 0,IState = 0;
    String ATR     = "" ,RdrName = "",PortName = "";
    long   AtrLen  = 0L,lOldState = 0L,lProto = 0L;
    boolean RetBool  = false;

    //Variables de regreso de estado

    int SLBAPI_OK = 11,SLBAPI_CARDABSENT = 2, SLBAPI_CARDSWALLOWED = 3,
       SLBAPI_CARDPOWERED = 4;

   //si el lector no está disponible, lo intentará leer nuevamente
    if (hReader == 0)
    {
        RetBool = SLBAPIAllocateComManager();
        RdrName = "REFLEX60"; //nombre del lector
        PortName = "COM1"; // puerto donde está conectado
        RetValue = SLBAPIAllocate(RdrName, PortName, hReader, IState);  //la llamada a este
                                                                        //método
      if (RetValue != SLBAPI_OK)                                  //intentará inicializar el lector
      {
        System.out.println("Error al asignar Lector");
        RetValue = SLBAPIFree(hReader, IState); //libera el puerto ya que no pudo asignar el lector
        RetBool = SLBAPIFreeComManager(); //libera el manejador de lector
        return false;
      }
     }

    //si la tarjeta esta ausente y el lector esta conectado bien permanecerá esperando hasta que
    //se introduzca la tarjeta
    if ( IState == SLBAPI_CARDABSENT)
    {
      lOldState = IState;
      RetValue = SLBAPIMonitor(hReader, lOldState, IState); //verifica el formato correcto de la tarjeta

      if (RetValue != SLBAPI_OK)
      {
```

```
        System.out.println("Error al insertar tarjeta");
        RetValue = SLBAPIFree(hReader, lState); //libera el puerto de la lectora. Tarjeta corrupta
        RetBool  = SLBAPIFreeComManager(); //libera el manejador del lector
        return false;
      }
    }

    AtrLen = 10;
    ATR = "0000000000";

    //Energiza la tarjeta
    if (lState == SLBAPI_CARDSWALLOWED)  //pregunta si la tarjeta no ha sido energizada
    {
      RetValue = SLBAPIPowerUp(hReader, lState); //este método energiza la tarjeta
      if (RetValue != SLBAPI_OK)
      {
        System.out.println("Error al energizar Tarjeta");
        RetValue = SLBAPIFree(hReader, lState);
        RetBool = SLBAPIFreeComManager();
        return false;
      }
      RetValue = SLBAPIState(hReader, lState, lProto, ATR, AtrLen); //devuelve el estado de la tarjeta
      if (RetValue != SLBAPI_OK)
      {
        System.out.println("Error al Aplicar Reset");
        return false;
      }
    }
    if (lState == SLBAPI_CARDPOWERED)
    {
      RetValue = SLBAPIReset(hReader, lState, ATR, AtrLen);   //le aplica un reset a la tarjeta
      if (RetValue != SLBAPI_OK)
      {
        System.out.println("Error al Aplicar Reset");
        return false;
      }
    }
    return true;
  }//metodo


//Devuelve la información contenida en la tarjeta
  public byte[] obtenDatos()
  {
    byte abyTexto = new byte[95];

    String stDatos = "00A5000002", stSW1, stSW2, Answer, CardAnswer;
    int inRes =
      SLBAPISendIsoOutT0(hReader, stDatos, (short)4406, lState, CardAnswer, stSW1, stSW2);

    Answer = stSW1 + stSW2;

    if (inRes == SLBAPI_OK)
    {
```

```
    if (Answer = "9000")
    {
        abyDatos = stDatos.getBytes();
        System.arrayCopy(abyDatos,0,abyTexto,0,113);
        return abyTexto;
    }
    else
        return null;
  }
  return null;
}

//Inserta el arreglo de bytes dentro de la tarjeta
public  boolean insertaDatos(byte[] datos)
{
  String stDatos = new String(datos, 0, datos.length), stSW1, stSW2, Answer;
  int inRes = SLBAPISendIsoInT0(hReader, stDatos, (short)(datos.length), lState, stSW1, stSW2);
  Answer = stSW1 + stSW2;

  if (inRes == SLBAPI_OK)
  {
    if (Answer = "9000")
    {
        return true;
    }
    else
        return false;
  }
  return false;
}

//se obtiene el arreglo de bytes que contiene la imagen de la tarjeta
public byte[] obtenFoto()
{
        byte abyTexto = new byte[4114];
        System.arrayCopy(abyDatos,131,abyTexto,0,4114);
        return abyTexto;

}
}
```

ANEXO 3

Programa del applet (Datos Generales del Afiliado)

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;


//Esta clase muestra  los datos existentes en la tarjeta inteligente

public class CAutentifica extends Applet implements ActionListener,FocusListener
{
 TextField          tfNombre,tfPaterno,tfMaterno,tfSuma,tfDeducible,tfCobertura,tfUltima;
 TextField          tfSangre,tfSocio,tfVigencia;
 Label              lbNombre,lbPaterno,lbMaterno,lbSuma,lbDeducible,lbCobertura,lbSangre;
 Label              lbUltima,lbSocio,lbVigencia;
 Button             btTraer,btCancelar,btEnviar;
 Canvas             caFoto;
 GridBagConstraints obGbc;
 GridBagLayout      obGbl;
 Panel              plCajas,plBoton;
 Font               foTitulo;
 Graphics           gaFoto;
 Image              imFoto;
 int                inBan = 1;

 public void init()
 {
  setLayout(new BorderLayout());
  caFoto  = new Canvas();
  caFoto.setSize(100,100);
  caFoto.setBackground(Color.white);
  foTitulo = new Font("Serif",Font.BOLD,16);
  obGbc   = new GridBagConstraints();
  obGbl   = new GridBagLayout();
  plCajas  = new Panel();
  plCajas.setLayout(obGbl);
  plBoton  = new Panel();

  tfNombre   = new TextField(15);
  tfPaterno  = new TextField(15);
  tfMaterno  = new TextField(15);
  tfSuma     = new TextField(10);
  tfDeducible = new TextField(10);
  tfCobertura = new TextField(10);
  tfUltima   = new TextField(10);
  tfSangre   = new TextField(8);
```

```
tfSocio    = new TextField(8);
tfVigencia  = new TextField(8);

tfCobertura.addFocusListener(this);

lbNombre    = new Label("Nombre: ");
lbPaterno   = new Label("A. Paterno: ");
lbMaterno   = new Label("A. Materno: "),
lbSuma      = new Label("Suma Asegurada: ");
lbDeducible = new Label("Deducible: ");
lbCobertura = new Label("Cobertura: ");
lbSangre    = new Label("Tipo de Sangre: ");
lbUltima    = new Label("Ultima visita al Medico: ");
lbSocio     = new Label("Socio desde: ");
lbVigencia  = new Label("Vigencia de la Poliza: ");

btCancelar  = new Button("Limpiar Campos");
btEnviar    = new Button("Actualizar Informacion");

btCancelar.addActionListener(this);
btEnviar.addActionListener(this);

tfNombre.setEnabled(false);
tfPaterno.setEnabled(false);
tfMaterno.setEnabled(false);
tfSocio.setEnabled(false);
tfSangre.setEnabled(false);

                    //comenzando pegado de componentes
obGbc.insets = new Insets(5,0,5,0);                    //arriba, izquierda, abajo, derecha
acomoda(lbNombre,0,0,GridBagConstraints.EAST); //etiqueta  Nombre
acomoda(tfNombre,1,0,GridBagConstraints.WEST); //Textfield Nombre
acomoda(lbPaterno,0,1,GridBagConstraints.EAST); //etiqueta  paterno
acomoda(tfPaterno,1,1,GridBagConstraints.WEST); //textfield paterno
acomoda(lbMaterno,0,2,GridBagConstraints.EAST); //etiqueta  materno
acomoda(tfMaterno,1,2,GridBagConstraints.WEST); //textfield materno
acomoda(lbSuma,0,3,GridBagConstraints.EAST);    //etiqueta  Suma
acomoda(tfSuma,1,3,GridBagConstraints.WEST);    //textfield monto
acomoda(lbDeducible,0,4,GridBagConstraints.EAST); //etiqueta  deducible
acomoda(tfDeducible,1,4,GridBagConstraints.WEST); //textfield deducible
acomoda(lbCobertura,0,5,GridBagConstraints.EAST); //etiqueta  cobertura
acomoda(tfCobertura,1,5,GridBagConstraints.WEST); //textfield cobertura
acomoda(lbSangre,2,3,GridBagConstraints.EAST);
acomoda(tfSangre,3,3,GridBagConstraints.WEST);
acomoda(lbUltima,2,4,GridBagConstraints.EAST);
acomoda(tfUltima,3,4,GridBagConstraints.WEST);
acomoda(new Label(""),0,6,GridBagConstraints.WEST);
acomoda(new Label(""),1,6,GridBagConstraints.WEST);
acomoda(new Label(""),2,6,GridBagConstraints.WEST);
acomoda(new Label(""),3,6,GridBagConstraints.WEST);
acomoda(lbSocio,0,8,GridBagConstraints.EAST);
acomoda(tfSocio,1,8,GridBagConstraints.WEST);
acomoda(lbVigencia,2,8,GridBagConstraints.EAST);
acomoda(tfVigencia,3,8,GridBagConstraints.WEST);
```

```
obGbc.insets = new Insets(5,30,10,0);//arriba, izquierda, abajo, derecha
obGbc.gridheight = 3;
obGbc.gridwidth  = 2;
acomoda(caFoto,2,0,GridBagConstraints.CENTER);  //foto

plBoton.add(btCancelar);plBoton.add(btEnviar);
add("Center",plCajas);
add("South",plBoton);
resize(653,463);

btEnviar.setEnabled(false);
btCancelar.setEnabled(false);

bytes();
tfNombre.setText   (mainDisplay.astDatos[0]);
tfPaterno.setText  (mainDisplay.astDatos[1]);
tfMaterno.setText  (mainDisplay.astDatos[2]);
tfSuma.setText     (mainDisplay.astDatos[3]);
tfDeducible.setText(mainDisplay.astDatos[4]);
tfCobertura.setText(mainDisplay.astDatos[5]);
tfUltima.setText(mainDisplay.astDatos[6]);
tfSangre.setText(mainDisplay.astDatos[7]);
tfSocio.setText(mainDisplay.astDatos[8]);
tfVigencia.setText(mainDisplay.astDatos[9]);

btEnviar.setEnabled(true);
btCancelar.setEnabled(true);

}//fin de init

public void paint(Graphics g)
{
 boolean boCargado = false;
 gaFoto = caFoto.getGraphics();

 while (!boCargado)
      boCargado = gaFoto.drawImage(imFoto,0,0,100,100,this);
}

/**Acomoda  los componentes de acuerdo
 *a los parametros recibidos. Nombre del componente,
 *columna, renglon, alineacion.
 */
public void acomoda(Component coSN,int inCol,int inRen,int inPos)
{
  obGbc.gridx  = inCol;  //columna
  obGbc.gridy  = inRen;  //renglon
  obGbc.anchor = inPos;  //alineacion de los componentes
  obGbl.setConstraints(coSN,obGbc);
  plCajas.add(coSN);
}//fin de acomoda
```

```
/**Metodo para pintar la imagen
 *en el canvas
 */
public void foto(byte aby[])
{
  gaFoto = caFoto.getGraphics();

  if(inBan == 1)
  {
    imFoto = caFoto.getToolkit().createImage(aby);
    inBan = 0;
  }
  repaint();
}


/**Metodo que lee  un arreglo de bytes
 *para despues colocarlo en el canvas
 */
public void bytes()
{
  aby = mainDisplay.bytesFoto();
  foto(aby);
}

public void focusGained(FocusEvent ke)
{
   btEnviar.setEnabled(true);
}

public void focusLost(FocusEvent ke) { }

public void actionPerformed(ActionEvent ae)
{
   if("Limpiar Campos".equals(ae.getActionCommand()))
   {
     tfSuma.setText("");
     tfDeducible.setText("");
     tfCobertura.setText("");
     tfUltima.setText("");
     tfVigencia.setText("");
     btEnviar.setEnabled(false);
   }
   else
   if("Actualizar Informacion".equals(ae.getActionCommand()))
   {
     if(!tfNombre.getText().equals("") && !tfPaterno.getText().equals("") &&
       !tfMaterno.getText().equals("") && !tfSuma.getText().equals("") &&
       !tfDeducible.getText().equals("") && !tfCobertura.getText().equals(""))
    {
      repaint();
      mainDisplay.astDatos[0] = tfNombre.getText().toUpperCase();
      mainDisplay.astDatos[1] = tfPaterno.getText().toUpperCase();
      mainDisplay.astDatos[2] = tfMaterno.getText().toUpperCase();
```

```
        mainDisplay.astDatos[3] = tfSuma.getText().toUpperCase();
        mainDisplay.astDatos[4] = tfDeducible.getText().toUpperCase();
        mainDisplay.astDatos[5] = tfCobertura.getText().toUpperCase();
        mainDisplay.astDatos[6] = tfUltima.getText().toUpperCase();
        mainDisplay.astDatos[7] = tfSangre.getText().toUpperCase();
        mainDisplay.astDatos[8] = tfSocio.getText().toUpperCase();
        mainDisplay.astDatos[9] = tfVigencia.getText().toUpperCase();


        String stTexto = mainDisplay.astDatos[0] + "|" + mainDisplay.astDatos[1] + "|" +
mainDisplay.astDatos[2] + "|" + mainDisplay.astDatos[3]+ "|" + mainDisplay.astDatos[4] + "|" +
mainDisplay.astDatos[5] + "|" + mainDisplay.astDatos[6] + "|" + mainDisplay.astDatos[7] + "|" +
mainDisplay.astDatos[8] + "|" + mainDisplay.astDatos[9];

          mainDisplay.inserta();

        new regresa(this,"vc/principal/autogen_mainDisplay.html"); //Llamada a la pagina que contiene
                                                                   //el menu principal

      }else
      {
        new CMensaje("E R R O R","Presiona \" Si \" para Llenar Datos Faltantes",this);
      }
    }

  }//fin de escuchador
}


//Pantalla que contiene el menu principal

import java.awt.*;
import java.applet.*;
import symantec.itools.awt.RadioButtonGroupPanel;
import symantec.itools.awt.shape.Rect;

public class mainDisplay extends Applet
{
      static boolean   obCarga    = false;
      static String    astDatos[] = null;
           CardletPC cardlet    = new CardletPC();

          public void init()
          {
                setLayout(null);
                setSize(601,337);
                setFont(new Font("Dialog", Font.PLAIN, 12));
                shRec = new symantec.itools.awt.shape.Rect();
                try {
                      shRec.setBevelStyle(symantec.itools.awt.shape.Rect.BEVEL_LOWERED);
                }catch(java.beans.PropertyVetoException e) { }
                shRec.setBounds(48,12,504,228);
                shRec.setFont(new Font("Dialog", Font.PLAIN, 12));
                add(shRec);
```

```
            plBoton = new java.awt.Panel();
            plBoton.setLayout(null);
            plBoton.setBounds(228,264,120,40);
            add(plBoton);
            btCarga = new java.awt.Button();
            btCarga.setActionCommand("button");
            btCarga.setLabel("Recargar Tarjeta");
            btCarga.setBounds(12,12,102,24);
            btCarga.setBackground(new Color(12632256));
            plBoton.add(btCarga);
            plCheck = new java.awt.Panel();
            plCheck.setLayout(null);
            plCheck.setBounds(72,36,468,180);
            add(plCheck);
            Group1 = new CheckboxGroup();
            ckDatos = new java.awt.Checkbox("Datos Generales", Group1, true);
            ckDatos.setBounds(60,12,132,12);
            plCheck.add(ckDatos);
            ckMedica = new java.awt.Checkbox("Prescripción Médica", Group1, false);
            ckMedica.setBounds(300,12,144,12);
            plCheck.add(ckMedica);
            ckFarmacia = new java.awt.Checkbox("Farmacia", Group1, false);
            ckFarmacia.setBounds(168,96,132,12);
            plCheck.add(ckFarmacia);
            plCheck.setEnabled(false);

            SymFocus aSymFocus = new SymFocus();
            ckDatos.addFocusListener(aSymFocus);
            ckMedica.addFocusListener(aSymFocus);
            ckFarmacia.addFocusListener(aSymFocus);
            SymAction lSymAction = new SymAction();
            btCarga.addActionListener(lSymAction);
    }

    symantec.itools.awt.shape.Rect shRec;
    java.awt.Panel plBoton;
    java.awt.Button btCarga;
    java.awt.Panel plCheck;
    java.awt.Checkbox ckDatos;
    CheckboxGroup Group1;
    java.awt.Checkbox ckMedica;
    java.awt.Checkbox ckFarmacia;

    //Muestra el frame de password
public void pass(String stUrl)
{
            if(mainDisplay.obCarga)
            {
              frPass obPass = new frPass(plCheck,this,stUrl);
              obPass.show();
              obPass.tamano();
            }
            else
            {
```

```
            new regresa(this,"vc/principal"+stUrl);
            }
}


    //Arma el arreglo de String a partir del arreglo de bytes
    public void armaArreglo(byte[] datos)
    {
       mainDisplay.astDatos[0] = new String(datos,18,33);//Nombre
       mainDisplay.astDatos[1] = new String(datos,33,48);//apellido P
       mainDisplay.astDatos[2] = new String(datos,48,63);//Apellido M
       mainDisplay.astDatos[3] = new String(datos,72,82);//Suma asegurada
       mainDisplay.astDatos[4] = new String(datos,82,87);//Deducible
       mainDisplay.astDatos[5] = new String(datos,87,97);//Cobertura
       mainDisplay.astDatos[6] = new String(datos,97,105);//Fecha de última visita al médico
       mainDisplay.astDatos[7] = new String(datos,71,72);//Tipo de sangre
       mainDisplay.astDatos[8] = new String(datos,105,113);//Socio desde
       mainDisplay.astDatos[9] = new String(datos,63,71);//Fecha Vencimiento de la poliza
    }

    //se obtiene el arreglo de bytes que contiene la imagen de la tarjeta
    public static byte[] bytesFoto()
    {
            return cardlet.obtenFoto();
    }


    //inserta los datos a la tarjeta
    public void inserta()
  {
    byte[] abyEnvio =
          {(byte)0x00,(byte)0xA4,(byte)0x00,(byte)0x00,(byte)0x02,(byte)0x22,(byte)0x22};
    }

    class SymFocus extends java.awt.event.FocusAdapter
    {
            public void focusGained(java.awt.event.FocusEvent event)
            {
                    Object object = event.getSource();
                    if (object == ckDatos)
                            ckDatos_GotFocus(event);
                    else if (object == ckMedica)
                            ckMedica_GotFocus(event);
                    else if (object == ckFarmacia)
                            ckFarmacia_GotFocus(event);
            }
    }

    void ckDatos_GotFocus(java.awt.event.FocusEvent event)
    {
       pass("/autogen_Datos.html");
    }
```

```
    void ckMedica_GotFocus(java.awt.event.FocusEvent event)
    {
       pass("/autogen_Medica.html");
    }

    void ckFarmacia_GotFocus(java.awt.event.FocusEvent event)
    {
       pass("/autogen_Farmacia.html");
    }

    class SymAction implements java.awt.event.ActionListener
    {
            public void actionPerformed(java.awt.event.ActionEvent event)
            {
                    Object object = event.getSource();
                    if (object == btCarga)
                            btCarga_Action(event);
            }
    }

    void btCarga_Action(java.awt.event.ActionEvent event)
    {
    CMensaje obMensaje = new CMensaje("A V I S O","¿ Se Inserto la Tarjeta ?",this);
    if(obMensaje.boOpcion)
    {
          if( cardlet.asignaLector() )
            {
                mainDisplay.obCarga = true;
                plCheck.setEnabled(true);
                byte[] dat = cardlet.obtenDatos();
                armaArreglo(dat);
            }
            else
            {
                mainDisplay.obCarga = false;
                plCheck.setEnabled(false);
            }
    }
    }
}
```

ANEXO 4

Esquema de la estructura de directorios de la Tarjeta inteligente

Files Description	ID	Type	Read&Seek	Data	N. Rec.	Rec Length	File Size	Header	Total Size
ROOT	3F00	DIR						24	24
EF_Key_ext	0011	TR	Never	Llaves ADM0,ADM1,ADM2			46	18	64
CHV1	0000	TR	Never	PIN de la tarjeta			23	18	41
ICC_NS	0002	TR	Always	Numero de serie			8	18	26
DATOS PER	7F10	DIR						24	24
Nombre	2F00	TR	ADM1	Nombre del asegurado	3	15	45	18	63
FecVen	2F00	TR	ADM1	Fecha vencimiento de póliza			8	0	8
Sangre	2F00	TR	ADM1	Tipo de Sangre			1	0	1
Suma	2F00	TR	ADM1	Suma Asegurada			10	0	10
Deducible	2F00	TR	ADM1	Deducible			5	0	5
Cobertura	2F00	TR	ADM1	Cobertura			10	0	10
FecUltVisita	2F00	TR	ADM1	Fecha ultima vesita al Médico			8	0	8
Socio	2F00	TR	ADM1	Socio desde			8	0	8
Foto	2F01	TR	ADM1	Fotografía del asegurado			4096	18	4114
									4406

ANEXO 5

Programa que utiliza los métodos y variables del API JavaCard FrameWork (JCF)

```
/* Este es un ejemplo de una aplicación ISO.
 * Este programa ejecuta un método por cada comando.
*/
import javacard.framework.*;  // Importación de paquetes y clases
import javacardx.framework.*;

                        // extiende a la clase _GPOS que contiene
                        // los métodos para iteractuar con el sistema operativo nativo de la tarjeta.
public class  sample extends _GPOS implements ST, ISO {

  // Constantes utilizadas a través del programa
  static final byte BUFFER_LENGTH    = 32;
  static final byte ACK_SIZE         = (byte)1;
  static final byte ACK_CODE         = (byte)0;
  static final byte OS_HEADER_SIZE   = (byte)0x10;
  static final byte GPOS_CREATE_FILE = (byte)0xE0;


  // Variables de clase
  public byte pbuffer[];
  public byte dbuffer[];
  public byte ackByte[];

  // inician los métodos
  public sample() {   //Constructor de la clase
```

```
  pbuffer  = new byte[ISO.COMMAND_LENGTH];
  dbuffer  = new byte[BUFFER_LENGTH];
  ackByte = new byte[ACK_SIZE];
 }



/** Método para seleccionar un archivo o directorio
 */
 public byte SelectFile() {
  byte status;  //aqui se almacena el SW regresado por los métodos llamados
  short fileId;

  // Se asume que la longitud siempre es 2
  if (pbuffer[4] != 2) {
   status = INVALID_PARAMETER;
  } else {

   // Se obtiene el nombre del archivo y el comando
   GetMessage(dbuffer,(byte)2,pbuffer[1]);

   // Hace un cast  de dbuffer a un short.
   fileId = (short) ((dbuffer[0] << 8) | (dbuffer[1] & 0x00FF));
   status = SelectFile(fileId);
  }

  return (status);
 }

/** Método que manda la ejecución al archivo indicado
  */
 public byte InitApplication() {
  byte status;
  short fileId;

  //  Debe enviar el id de un archivo válido
```

```
  GetMessage( dbuffer,(byte)1,pbuffer[1]);
  fileId = (short)((pbuffer[2] << 8) | (pbuffer[3] & 0x00FF));
  status = Execute(fileId, dbuffer[0]);  //Manda la ejecución al archivo seleccionado
  return (status);
 }


/**Crea un archivo
 */
 public byte CreateFile() {
  byte status;

  if (pbuffer[4] != OS_HEADER_SIZE) {
   status = INVALID_PARAMETER;
  } else {

   // Recibe el comando y nombre del archivo
   GetMessage( dbuffer,pbuffer[4],pbuffer[1]);
   status = CreateFile(dbuffer);
  }
  return (status);
 }

/** Lee el archivo seleccionado
 */
 public byte ReadBinary() {
  byte status;
  short fileId;
  fileId = (short)((pbuffer[2] << 8) | (pbuffer[3] & 0x00FF));

  // Asume que el archivo ya esta seleccionado
  status = ReadBinaryFile (fileId, pbuffer[4], dbuffer);

  // Envía los datos si fue exitosa la lectura
  ackByte[0] = pbuffer[1];
  if (status == SUCCESS) {
```

```
    SendMessage(ackByte,ACK_SIZE);  //envia datos de confirmación de regreso de datos
    SendMessage(dbuffer,pbuffer[4]);    //envía datos
   }

   return(status);
  }

  // despachador
  public byte Process(byte c) {

   switch (c) {
    case ISO.SELECT_FILE:       return (SelectFile());
    case ISO.INIT_APPLICATION:   return (InitApplication());
    case GPOS_CREATE_FILE:       return (CreateFile());
    case ISO.READ_BINARY:       return (ReadBinary());
    default:                    return (INS_NOT_SUPPORTED);
   }
  }

 //Método que será llamado cuando se corra el programa
 public static void main (String args[]) {

    // Inicializa el regreso de memoria a este programa
    Execute((short)0, (byte)0);

    byte bReturnStatus;
    byte command;
    sample b = new sample();
    do {
     // Obtiene el primer comando
     GetMessage(b.pbuffer, ISO.COMMAND_LENGTH, ACK_CODE);

     // Verifica el formato del mensaje
       if ((b.pbuffer[0] != ISO.APP_CLASS) && (b.pbuffer[0] != ISO.CLASS)) {
            bReturnStatus = INVALID_CLASS;
       } else {
```

```
      // Verifica si la longitud de los datos es muy larga
      if (b.pbuffer[4] > BUFFER_LENGTH) {
        bReturnStatus = INVALID_PARAMETER;
        } else {
        // Procesa el comando leído
        command = b.pbuffer[1];
        bReturnStatus = b.Process(command);
        }
    }
          // Inicializa el SW a enviar, basado en el resultado del método Process
    SendStatus(bReturnStatus);
          // Envía un SW a la terminal
          SendSW1SW2();
  } while (true);
 }
}
```

GLOSARIO

API.- Aplication Programming Interface. Es decir, la interfaz de programación de aplicaciones. Conjunto de estándares y convenciones mediante los cuales los programas pueden llamar servicios de red o sistemas operativos específicos.

Applet.- Es un programa dinámico e interactivo que se puede ejecutar dentro de una página Web. También conocido como mini – aplicación o aplicación ligera.

ASCII.- American Standard Code for Information Interchange. Es decir, el Estándar Americano para Intercambio de Información. La tabla básica de caracteres ASCII está compuesta por 128 caracteres incluyendo símbolos y caracteres de control. Existe una versión extendida de 256.

ATR.- Answer to Reset. Es decir, la Respuesta de Inicialización. Es la iniciación del cardlet en la JavaCard.

Autenticación – Autentificación.- Autorizar o Legalizar algo.

Bit.- Dígito Binario. Unidad mínima de información, puede tener dos estados "0" o "1".

Browser.- Explorador, Navegador. Programa que permite al usuario Navegar por el Web.

Byte.- El tamaño de un caracter (8 bits).

Bytecode.- Código generado al compilar un programa en Java, almacenado en un archivo con extensión .class

Cardlets.- Programas que se ejecutan dentro de la tarjeta inteligente.

Casting.- Cambiar de propiedades al objeto o tipo de dato primitivo.

Chip.- Pequeño circuito integrado de la memoria de un procesador.

CPU.- Central Processing Unit. Es decir, Unidad Central de Procesamiento. Circuitería interna de control, procesamiento y almacenamiento de la computadora, que incluye a la unidad de aritmética y lógica (ALU), a la unidad de control, a la memoria de sólo lectura (ROM) y a la memoria de acceso aleatorio (RAM).

Criptográfico.- Mensaje cifrado de manera que no pueda leerlo alguna otra persona distinta al destinatario designado.

Demonio.- Programa que se ejecuta y corre de manera infinita en background en alguna máquina (computadora), y que siempre está escuchando peticiones de los clientes que se conectan a esta. Por ejemplo, existe el demonio de telnet, el cual está siempre ejecutándose y permitiendo abrir una sesión a cada cliente que se conecta a la máquina (computadora) en la cual se está corriendo ese programa.

DF.- Dedicated File. Es decir, Archivo Dedicado.

EEPROM.- Electrically Erasable Programmable Read Only Memory. Es decir, Memoria de Sólo Lectura, borrable y programable eléctricamente. Tipo de memoria de sólo lectura (ROM) que puede ser borrada y programada aplicando una corriente eléctrica a los chips que la contiene y escribiendo nuevas instrucciones en ellos.

EF.- Elementary File. Es decir, Archivo Elemental.

Encriptar.- En Criptografía, proceso de convertir un mensaje en texto cifrado (encriptado) utilizando una clave, de manera que parezca que el mensaje sólo contiene basura.

EPROM.- Erasable Programmable Read Only Memory. Es decir, Memoria Programable y Borrable de Sólo Lectura. Chip de memoria de sólo lectura (ROM) que puede ser programado y reprogramado con un dispositivo electrónico especial.

Evento.- Acontecimiento, suceso ocurrido dentro de un Objeto gráfico de Java.

Excepción.- Son los eventos no deseados que pueden ocurrir en el tiempo de ejecución de un programa.

File System.- Sistema de Archivos. Estructura jerárquica de ubicación de archivos y directorios.

Frame.- Ventana de alto nivel con título.

FTP.- File Transfer Protocol. Es decir, es el Protocolo de transferencia de archivos. Estándar de Internet para intercambio de archivos.

GSM.- Global System for Mobile comunications. Es decir, el sistema global para comunicaciones móviles.

Hardware.- Componentes electrónicos, tarjetas, periféricos y equipo que conforman un sistema de computación.

HTML.- HyperText Markup Language. Es decir, Lenguaje de Marcas de Hypertexto. Lenguaje para elaborar páginas Web.

HTTP.- Hypertext Tranfer Protocol. Es decir, el Protocolo de Transferencia de Hypertexto.

Importar.- Utilizar las propiedades del objeto a heredar.

JavaCard.- Tarjeta Inteligente para programar con Java.

JDK.- Java Development Kit. Es decir, el paquete de utilerías y herramientas de desarrollo para Java, creado por Sun Microsystems y distribuido sin costo.

JVM.- Java Virtual Machine. Es decir, el intérprete y ambiente de ejecución de Java. También conocido como máquina virtual, porque sin importar el tipo de computadora donde se esté ejecutando crea una computadora simulada que proporciona la plataforma correcta para ejecutar programas de Java.

MF.- Master File. Es decir, Archivo Maestro.

Multihilos (Multithreads).- Procesos que pueden ejecutarse al mismo tiempo.

NIP.- Número de Identificación Personal. Por sus siglas en inglés (PIN) se le conoce también como Personal Identification Number.

Password.- Conjunto de caracteres que identifican un acceso a una aplicación.

Plataforma.- Es el Sistema Operativo bajo el cual se ejecutan las aplicaciones dentro de la computadora.

PROM.- Programmable Read Only Memory. Es decir, Memoria Programable de Sólo Lectura. Chip de memoria de sólo lectura (ROM) programado de fábrica para emplearlo con una computadora determinada. A diferencia de los chips de ROM estándar, que tienen la programación integrada en el diseño interno de sus circuitos, los chips de ROM programables son fáciles de modificar.

RAM.- Random Access Memory. Es decir, Memoria de Acceso Aleatorio. Memoria principal de trabajo de una computadora, en la cual se guardan instrucciones y datos de programas para que la Unidad Central de Procesamiento (CPU) pueda acceder a ellos directamente a través del bus de datos externo de alta velocidad.

Recolector de Basura.- Proceso por el cual una programa pasa a través de la Memoria de Acceso Aleatorio (RAM), decide cual información almacenada ahí ya no es útil y prepara las direcciones que la contienen para su reutilización.

ROM.- Read Only Memory. Es decir, Memoria de Solo Lectura. Parte del almacenamiento principal de una computadora que no pierde su contenido cuando se interrumpe el flujo de energía eléctrica. Contiene programas esenciales de sistema que ni el usuario ni la computadora pueden borrar.

Socket.- Es el punto final de la conexión entre dos máquinas (computadoras).

Software.- Es un conjunto de programas que se ejecutan dentro de la computadora.

TCP / IP.- Transmission Control Protocol / Internet Protocol. Es decir, protocolo de control de transmisión / protocolo Internet. También es una frase utilizada normalmente para referirse a todo el grupo de protocolos de Internet.

Web.- Red Mundial de Comunicación.

WebServer.- (Hardware) Es una computadora, denominada servidor, a partir de la cual las computadoras cliente se conectan para poder ejecutar los programas residentes en dicho servidor. (Software) Es el software que permite compartir y administrar directorios, recursos del servidor, aplicaciones, páginas html, etc. . .

Zona Militarizada.- Es la zona en la cual, para accesar a ella se tienen ciertas restricciones. Como zona se debe de entender algún segmento de red.

Zona Desmilitarizada.- Es la zona en la cual para accesar a ella se puede hacer sin alguna restricción. Como zona se debe de entender algún segmento de red.