Aprendiendo Java

y

Programación Orientada a Objetos

Gustavo Guillermo Pérez

www.compunauta.com
gustavo@compunauta.com

*Hasta que esta leyenda no desaparezca el libro no ha sido terminado, revisar:
http://compunauta.com/forums/linux/programacion/java/ebook.html
para las actualizaciones diarias o semanales y para descargar el código de los ejemplos.
ESTE LIBRO ES LIBRE Y GRATUITO, CONSIDERA HACER UNA PEQUEÑA DONACIÓN EN LA WEB INDICADA ARRIBA.

*Para navegar el Índice con Adobe Acrobat Reader, abrir favoritos o bookmarks
Ultima actualización, lunes 11 de febrero de 2008

INDICE DEL TEMARIO

Prólogo

Este libro se hizo con la idea de proveer a mis alumnos y cursos un material didáctico con qué trabajar (también es útil para profesores y otros cursos) y al mismo tiempo dar una idea concisa de lo que significan los objetos en Java. La programación orientada a objetos en Java no es tratada y enfocada en muchos textos tutoriales, manuales o libros sobre el tema y considero que es la etapa inicial del aprendizaje donde se falla en muchos cursos respecto a Java. Por otro lado los estudiantes que recién se inician en la programación, no podrán ir muy lejos si no se les inculca los conceptos básicos, entonces este material no avanzara de lleno sobre la programación orientada a objetos, sino hasta que los conceptos de programación básicos, como tomas de decisión, bucles, variables etc, estén fijados lo suficiente.

Java es un lenguaje muy útil debido a la opción multiplataforma que provee (desde PC, Linux, Windows, hasta MAC, teléfonos, pocket PCs, etc.) y veremos en el transcurso de este texto como se puede optimizar la ejecución de una aplicación Java para que se aproxime a la ejecución de un binario nativo como los que se compilan con gcc.

Por lo tanto suponer que el alumno posee conocimientos avanzados de programación orientada a objetos o de otros lenguajes de programación no siempre suele ser la mejor opción, en este libro dejaremos en claro que el lenguaje de programación Java tiene sus operadores matemáticos básicos y tipos de datos básicos sin importar que estos mismos existan o no en otros lenguajes de programación.

Las sugerencias son bienvenidas.

I. TEMAS INTRODUCTORIOS

La Máquina Virtual de Java (JVM, Java Virtual Machine)

La máquina virtual de Java se denomina al *procesador o entorno virtual* que se utiliza para interpretar los bytecodes de los binarios de Java, ya que como sabemos Java se hizo para correr en cualquier plataforma sin recompilar los binarios. De esta manera este entorno virtual se puede obtener para nuestra arquitectura y sistema operativo sin modificaciones a nuestro programa original (esto no es cierto si utilizamos una mala dinámica de programación).

Podemos entonces generar un binario y este podrá Correr en Linux, MAC OSX, FreeBSD, Solaris, o Windows, y para las arquitecturas disponibles en las que podamos obtener la JVM, como ser AMD64, SPARC, PIV, etc. etc.

La máquina virtual de Java ha tenido la característica de ser un entorno de ejecución *pesado* en términos de recursos del procesador y memoria, que por medio de una administración rigurosa del sistema operativo estos podrían llegar a ser insuficientes y las aplicaciones ejecutarse de manera muy lenta. Esto no es cierto en la actualidad, existen alternativas a la JVM provista por Sun Microsystems que permiten una velocidad comparable a una aplicación compilada en C++ nativa en la arquitectura, un ejemplo de esto es Kaffe, Kaffe (www.kaffe.org) es una máquina de Java OpenSource que puede compilarse sin mayores modificaciones en nuestra arquitectura necesaria y correrá increíblemente más rápida que la distribución estándar de JVM de Sun Microsystems y consumirá muchos menos recursos.

Kit de desarrollo y Entorno de ejecución (JDK, JRE)

El Kit de desarrollo conocido como JDK (Java Development Kit) provee de un compilador, un mecanismo para comprimir un proyecto en un solo archivo de tipo JAR (que es compatible con ZIP) y un entorno de ejecución para nuestros binarios.

Cuando nuestro proyecto terminado se prepara para distribuir, no es necesario tener el compilador y la mayoría de las herramientas que se proveen en el JDK, entonces podemos prescindir de dicho JDK y utilizar el entorno de ejecución que es más pequeño en cuestiones sólo de espacio en disco. Este JRE (Java Runtime Environment) también puede redistribuirse sin problemas de licencias.

En las plataformas basadas en Linux, existen mejores herramientas de desarrollo y por supuesto en casi todas las distribuciones de Linux, se encuentra disponible Kit de desarrollo o JDK. Así como recomendamos Kaffe como JVM para proyectos avanzados o de alto rendimiento, recomendamos Jikes como compilador de ByteCodes para Java, Jikes no es interpretado como el compilador proporcionado en el JDK de Sun Microsystems.

Comandos que inician el JDK, JRE

Comando	*Descripción*
`java`	Inicia el entorno de ejecución recibiendo como argumento el nombre del binario ejecutable en formato ByteCodes sin la extensión de archivo *.class* que identifica de manera visual un binario java. Este comando es parte de JDK y JRE
`javac`	Inicia el compilador Java recibiendo como argumento todos los archivos de código fuente cuya terminación es *.java* incluida dicha extensión. Este comando no es parte de JRE.
`jar`	Por medio de este comando iniciamos el empaquetador de clases y archivos de Java que nos permiten fabricar un único archivo contenedor de nuestras aplicaciones, multimedia y gráficos. Este comando es parte sólo de JDK.

Salida por pantalla de los comandos del JDK, JRE

Si abrimos una consola de comandos, y ejecutamos estos comandos podremos detectar la versión del entorno de ejecución y las órdenes de entrada que estos soportan.

En Linux, podemos abrir una XTerm, buscando el menú ejecutar y escribiendo *xterm*. En Windows, podemos hacerlo abriendo el diálogo ejecutar y escribiendo *command* o *cmd* dependiendo si el sistema es basado en NT o 9X.

```
gus@gusgus ~ $ java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)

where options include:
    -client       to select the "client" VM
    -server       to select the "server" VM
    -hotspot      is a synonym for the "client" VM  [deprecated]
                  The default VM is client.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose[:class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -jre-no-restrict-search
                  include/exclude user private JREs in the version search
    -? -help      print this help message
    -X            print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
gus@gusgus ~ $ java -version
java version "1.4.2-02"
Java(TM) 2 Runtime Environment, Standard Edition (build Blackdown-1.4.2-02)
Java HotSpot(TM) Client VM (build Blackdown-1.4.2-02, mixed mode)
gus@gusgus ~ $
```

```
gus@gusgus ~ $ java -version
java full version "kaffe-1.4.2"

kaffe VM "1.1.6"

Copyright (c) 1996-2005 Kaffe.org project contributors (please see
  the source code for a full list of contributors).  All rights reserved.
Portions Copyright (c) 1996-2002 Transvirtual Technologies, Inc.

The Kaffe virtual machine is free software, licensed under the terms of
the GNU General Public License.  Kaffe.org is a an independent, free software
community project, not directly affiliated with Transvirtual Technologies,
Inc.  Kaffe is a Trademark of Transvirtual Technologies, Inc.  Kaffe comes
with ABSOLUTELY NO WARRANTY.

Engine: Just-in-time v3   Version: 1.1.6   Java Version: 1.4
Heap defaults: minimum size: 5 MB, maximum size: unlimited
Stack default size: 256 KB
```

```
gus@gusgus ~ $ javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
   -deprecation                 Output source locations where deprecated APIs are
used
  -classpath <path>          Specify where to find user class files
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -d <directory>             Specify where to place generated class files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -help                      Print a synopsis of standard options

gus@gusgus ~ $
```

```
gus@gusgus ~ $ jar
Sintaxis:   jar   {ctxu}[vfm0Mi]   [archivo-jar]   [archivo-manifest]   [-C   dir]
archivos ...
Opciones:
    -c  crear nuevo contenedor
    -t  mostrar contenido de contenedor
    -x  extraer archivos nombrados (o todos) del contenedor
    -u  actualizar contenedor existente
    -v  generar salida detallada en salida estándar
    -f  especificar nombre de archivo contenedor
    -m  incluir información de manifest del archivo manifest especificado
    -0  solo almacenar; no utilizar compresión ZIP
    -M  no crear un archivo manifest para las entradas
    -i  generar información de índice para los archivos jar especificados
    -C  cambiar al directorio especificado e incluir el archivo siguiente
Si alguno de los archivos es un directorio, se procesará de forma recursiva.
Se deben especificar los nombres del archivo manifest y del archivo contenedor
en el mismo orden en que se especifiquen los indicadores 'm' y 'f'.

Ejemplo  1:  para  archivar  dos  archivos  de  clase  en  un  contenedor  llamado
classes.jar:
       jar cvf classes.jar Foo.class Bar.class
Ejemplo 2: utilizar un archivo manifest existente, 'mymanifest', y archivar todos
los
           archivos del directorio foo/ en 'classes.jar':
       jar cvfm classes.jar mymanifest -C foo/ .

gus@gusgus ~ $
```

Como es sabido hay muchos lenguajes de programación orientada a objetos *POO* que tienen muchas similitudes entre si, pero puntualmente nos enfocaremos en Java, utilizaremos para tal efecto un modelo de fábrica de objetos, para introducir los términos *clase*, *objeto, método, propiedad, estático, dinámico,* donde la fábrica de objetos será el entorno de ejecución o JRE.

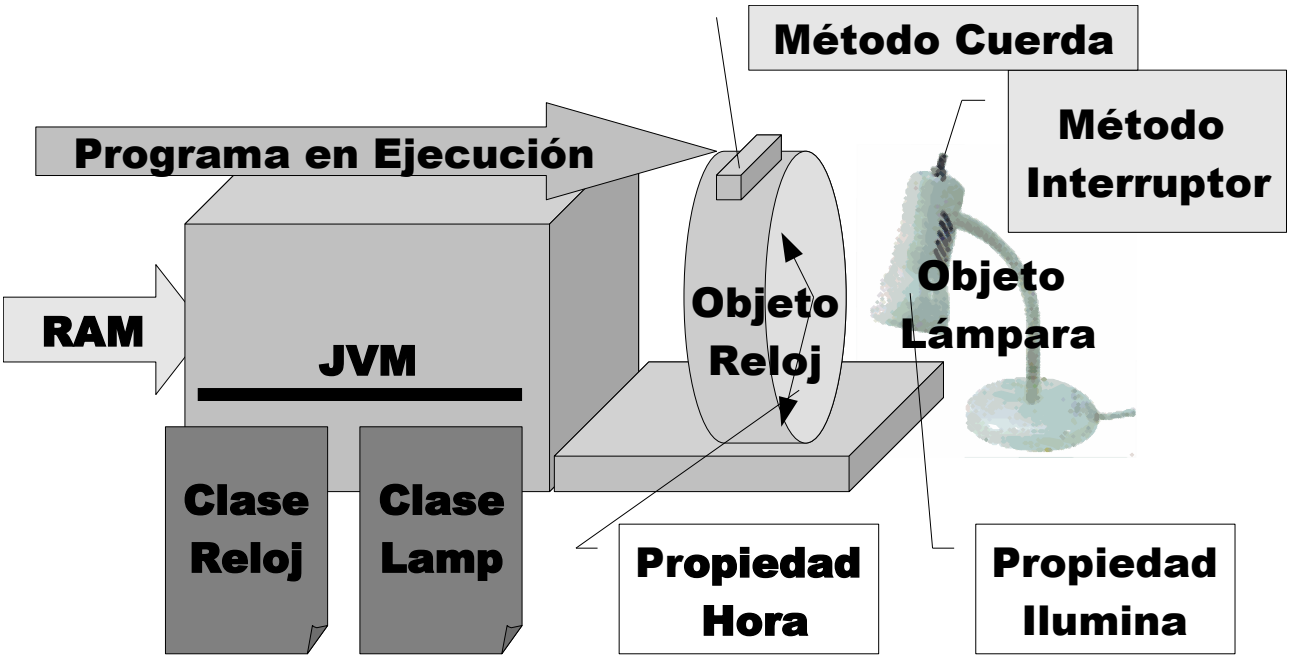

Ilustración 1: La fábrica de Objetos dentro de JRE

Descripción Empírica del funcionamiento del entorno JRE

Como podemos ver en la imagen anterior la fábrica de objetos utiliza memoria del sistema para fabricar los objetos, pero para fabricarlos necesita la información pregrabada llamada *clases*, las clases son las que almacenan las propiedades y métodos que contendrá un *objeto*. Un objeto cambiará sus propiedades o las propiedades de otros *objetos* por medio de los *métodos*. Los *métodos* que sólo pueden ejecutarse cuando el *objeto* existe, son los denominados dinámicos, y los métodos que pertenecen a la *clase* son denominados estáticos, porque pueden ser llamados sin necesidad de la existencia de un *objeto*.

En el ejemplo tenemos dos *objetos*, Reloj y Lámpara los cuales tienen *métodos* Cuerda e Interruptor, que son los que cambiaran el estado de las *propiedades* Hora e Iluminando. La clase reloj, podría tener un *método estático* llamado poner_en_hora, para ajustar todos los relojes fabricados. El programa en ejecución inicia en un *método estático* ya que no existen inicialmente objetos en nuestro programa, este *método* se llama *main* que se define como:

```
static int main(String[] args){
.....
}
```

Donde *args* es un vector o arreglo unidimensional que contendrá los argumentos que hayan sido pasados al comando java después de las órdenes y el nombre de la *clase* principal que alojará este *método*. El entorno de ejecución o la JVM fabricará objetos de sistema que podrán ser utilizados llamando a *métodos estáticos* de clases pertenecientes a la biblioteca de java.

Nota: **Java provee una documentación completa en inglés de toda la biblioteca incluida en la versión de JRE o JDK, una buena práctica es no utilizar las más avanzadas y complejas funciones de la API (Application Programing Interface) ya que al querer iniciar nuestro programa en una versión anterior que no tenga una biblioteca actualizada, se pueden producir problemas de métodos faltantes.**

Herencia soportada por Java

Java no soporta herencia múltiple, es decir no podemos fabricar un objeto más complejo con dos diferentes más simples, sino que sólo podremos heredar objetos nuevos de un sólo objeto padre, que proveerá los métodos y propiedades básicas que serán extendidas y/o ampliadas por el nuevo objeto. Es decir no podríamos tener un objeto Lámpara Reloj que derive del objeto Lámpara y del objeto Reloj, sino que tendríamos que usar otros mecanismos para proporcionar esa funcionalidad, interfaces y alojamiento de objetos.

Esto podría comprenderse como:

- Interfaces: Permiten que sean implementadas por objetos para adquirir comportamiento, pero el comportamiento no es provisto por la interfaz, sino que el programador debe proporcionar una manera eficaz de construir los métodos definidos en dicha interfaz uno por uno. Pueden implementarse varias interfaces al mismo tiempo, en todos los casos es necesario codificar funciones o métodos.
- Alojamiento: Podemos pensar que al objeto Lámpara le insertamos en su interior un objeto Reloj entonces podemos llamar a los métodos del Reloj que está en la Lámpara, esto lo veremos más adelante en algunos ejemplos. Ya no sería la Lámpara Reloj, pero sería una Lámpara **con** Reloj.
- Herencia: Nos permite crear un objeto nuevo en base a uno existente, es una nueva clase de objeto, puede utilizar el alojamiento de otros objetos como propiedades para adquirir funcionalidad.

Tipos de datos Básicos

En todos los lenguajes de programación existen tipos de datos básicos con los que se realizarán las operaciones matemáticas básicas y las operaciones booleanas de verdadero o falso.

Tipo de Datos	***Alcance o Rango***
int	de -2147483648 a 2147483647 (4bytes)
byte	de -128 a 127 (1Byete)
short	de -32768 a 32767 (2Bytes)
long	de -9223372036854775808 a 9223372036854775807 (8Bytes)
char	de '\u0000' a '\uffff', ambos incluidos que es lo mismo que de 0 a 65535, 1 letra
boolean	0 o 1 (1bit)
float	4Bytes, punto flotante
double	8Bytes, punto flotante
String	No es un tipo de datos básico, es un objeto básico, con propiedades y métodos, pero el lenguaje Java permite definir un nuevo objeto con el delimitador (“), por lo que podemos concatenar (unir) texto utilizando el operador (+) con los nombres de los objetos de tipo String y los trozos de texto delimitados con (“).

Tabla 1: Tipos de datos básicos

Operadores y Delimitadores Básicos

Operador	*Datos*	*Acción*
+	String	Une texto (concatenador).
	número	Suma.
–	número	invierte el signo del número.
	número	Resta.
*	número	Multiplica y tiene prioridad sobre la suma y resta.
/	número	Divide y tiene prioridad sobre la suma y resta.
%	número	Devuelve el resto de la división del operador derecho por el izquierdo.
!	booleano	El operador `NOT` booleano.
~	número	Invierte bit por bit el operando de la derecha
&	entero o booleano	El operador `AND`, booleano bit por bit del elemento a la izquierda contra el de la derecha.
&&	Cualquier	El operador AND condicional.
|	entero o booleano	El operador `OR`, booleano bit por bit del elemento a la izquierda contra el de la derecha.
||	Cualquier	El operador OR condicional.
<<	número	Desplaza los bits hacia la izquierda la cantidad de la derecha.
>>	número	Desplaza los bits hacia la derecha la cantidad de la derecha.
>>>	número	Desplaza bits hacia la derecha pero no conserva el signo.
==	Cualquier	El operador condicional de igualdad, devuelve verdadero si derecha e izquierda es lo mismo.
<	Básicos	Menor que, devuelve verdadero si izquierda es menor que derecha.
<=	Básicos	Menor o igual que, devuelve verdadero si izquierda es menor o igual que derecha.
>	Básicos	Mayor que, devuelve verdadero si izquierda es mayor que derecha.
>=	Básicos	Mayor o igual que, devuelve verdadero si izquierda es mayor o igual que derecha.
=	Cualquier	Asigna al elemento de la izquierda, el de la derecha.
`(op)=`	Básicos	Hace la operación op [+,-,*,/.etc..] entre los dos operandos y el resultado lo guarda en el de la izquierda.
var++	Básicos	Incrementa en uno la variable var.
var--	Básicos	Decrementa en uno la variable var.
.	objetos	Separa los nombres de los objetos o propiedades o métodos o nombres y jerarquía de clases.
`(int)`	tipo de datos	Convierte un tipo de datos a entero, si reemplazamos int por una clase de objetos, podemos siempre y cuando sea correcto convertir al tipo de datos indicado entre paréntesis.
;	Cualquier	Termina una línea de orden.
,	Cualquier	Separa argumentos de un método o función.
//	Cualquier	Empieza una línea de comentario.
/* */	Cualquier	Delimitan un trozo de texto como comentario.
/** */	Cualquier	Delimitan un trozo de texto como documentación para JavaDoc

Operador	*Datos*	*Acción*
{}	Cualquier	Delimitan un bloque de código de una estructura de datos.
( )	Cualquier	Asocia una operación teniendo prioridad, o en caso de un método agrupa los argumentos separados por comas.

Tabla 2: Operadores Básicos

Nombres de variables y funciones.

En Java los nombres de las variables y las funciones siguen ciertas reglas, por ejemplo las variables se acostumbra a nombrarlas con todas sus letras en minúsculas, las funciones con la primer palabra que la identifica en minúsculas y las palabras siguientes con la primer letra en mayúsculas y los nombres de las Clases de objetos con la primer letra de cada palabra que la define en mayúsculas.

Ejemplos:

Variables

```
int soles=3;
char salir='q';
```

Funciones:

```
public static int redondearPromedio(int[] valores){
...
}
```

Clases:

```
public class RelojAtomico{
....
}
```

II. Codificación Inicial y Estructuras de Datos

> ***Nota:*** ***En este capítulo veremos como codificar y compilar el primer programa, a continuación emplearemos los operadores y tipos de datos básicos y por último veremos las estructuras de datos básicas soportadas por Java.***

El primer programa

Suponemos que nuestro sistema operativo ya tiene instalado el kit de desarrollo (JDK) y procedemos a abrir un editor de texto de nuestro gusto. En KDE, podemos abrir ***kwrite*** en otros sistemas busquemos ***block de notas*** u otro editor como ***EditPlus***, de todas maneras cualquier editor estará bien.

Dentro del editor de textos escribiremos lo siguiente:

```
public class HolaMundo {
  public static void main(String[] args) {
    System.out.println("Hola, mi primer programa");
  }
}
```

El archivo debe guardarse como *HolaMundo.java* respetando las mayúsculas y minúsculas, necesitamos compilar el programa para obtener el binario con extensión class, entonces procedemos a llamar al compilador con una consola de comandos para poder después usar la JVM (comando java) para correr el programa, pero para trabajar mejor haremos una carpeta donde guardaremos los ejemplos, para los que usen Windows háganla en la unidad c:\ y nómbrenla ejemplos, para los que usamos un sistema mejor, en nuestra carpeta de usuario y la llamamos ejemplos también sólo que en minúsculas.

```
gus@gusgus ~ $ cd ejemplos
gus@gusgus ~/ejemplos $ javac HolaMundo.java
gus@gusgus ~/ejemplos $ java HolaMundo
Hola, mi primer programa
gus@gusgus ~/ejemplos $
```

> ***Nota:*** ***Más adelante veremos como utilizar un editor basado en Java para practicar este curso, antes de explicar línea por línea del programa, es necesario introducir un tema que será utilizado con mucha frecuencia, estos son los paquetes, los cuales darán el concepto de Clase Pública.***

Paquetes

En el ejemplo anterior hemos omitido una parte esencial de la programación en el lenguaje Java, estos son los *Paquetes*, se utilizan para diferenciar una clase respecto de otras con el mismo nombre, si nos ponemos a pensar sobre la cantidad de nombres diferentes de clases que podemos poner a los tipos de objetos en nuestros programas es infinita, pero dos programadores diferentes pueden llegar a usar dos veces el mismo nombre para una misma clase, de esta manera podemos hacer que sólo las clases de objetos que haremos visibles a otros programadores o que ejecutarán el componente principal del programa (el método estático main) sean públicas.

Las clases de objetos que no queramos hacer visibles a otros componentes de nuestro proyecto, entonces se podrán declarar como privadas, de esa manera sólo una clase de objetos que pertenezca al grupo de clases podrá acceder a las demás privadas.

Los paquetes suelen tener un nombre especial referente al de la organización que realiza el

proyecto, esto es una sugerencia, en realidad se puede utilizar casi cualquier nombre, así que por ejemplo si nuestra empresa es compunauta, y el nombre de dominio de los servicios que se proporcionan en Internet es *compunauta.com*, entonces nuestro paquete podría llamarse *com.compunauta* y a su vez si tenemos un subproyecto, este podríamos nombrarlo como *com.compunauta.aprendiendojava*, sólo tengamos en cuenta que es el orden inverso de como aparecería en Internet. Usamos puntos para separar los nombres y tenemos cuidado respecto de las mayúsculas y minúsculas.

> ***Nota: El lenguaje Java es sensible a las mayúsculas y minúsculas, es decir si tenemos el nombre del paquete com.compunauta.aprendiendojava no será lo mismo si lo escribimos como com.compunauta.AprendiendoJava. De la misma manera no es lo mismo HolaMundo que holamundo.***

Palabras clave o reservadas

Las palabras clave o reservadas son aquellas que definen o se combinan con una definición de datos de una variable o propiedad, clase de objetos o métodos, por ejemplo c*lass* es una palabra clave y nos indica que a continuación pondremos el nombre a la clase de objeto que vamos a crear, si queremos que dicha clase pueda ser accedida desde dentro o fuera de nuestro paquete de clases de objetos, entonces la definiremos como *public* (pública), donde esta es otra palabra clave.

A veces las palabras claves son palabras reservadas, por ejemplo *goto* es una palabra clave reservada en Java, ya que no tenemos permitido saltar a una línea en particular de nuestro programa, tenemos que trabajar con llamadas a funciones y otras maneras de codificación.

Tipos de datos

Definen tipos de datos que devolverán los métodos o los tipos de datos que definirán una propiedad básica. Estos ya los habíamos visto anteriormente pero ahora los describiremos por el espacio que ocupan en memoria.

Palabra Clave	*Significado*
`boolean`	Valores Booleanos (1bit*)
`byte`	1Byte, número entero (8bits)
`short`	2Bytes, número entero
`int`	4Bytes, número entero
`long`	8Bytes, número entero
`char`	2Bytes, (short) 1Caracter
`float`	4Bytes, punto flotante.
`double`	8Bytes, punto flotante, doble precisión
`void`	Cuando no hay nada que devolver
`null`	Tipo de datos Nulo, vacío, cualquier operación sobre null produce un error.

Tabla 3: Palabras claves - Tipos de datos

Permisos y declaración de clases, propiedades o métodos.

Definen los permisos a un método o clase de objetos, haciendo visible u oculto un método propiedad o clase a otras clases de objetos u otros objetos que quieran accederlas.

Palabra clave	*Significado*
public	Para una clase, que es accesible desde cualquier parte, para un método, que es accesible por cualquier método que pueda acceder a la clase de objetos. Para que una clase que deriva de otra tiene que ser pública la clase de objetos padre para poder ser accecible.
private	Sólo se permite a la misma clase de objetos acceder a sus propiedades o métodos.
protected	Sólo se permite al paquete acceder a esta clase.
package	Define a que paquete pertenece una clase.
import	Define que paquetes usaremos, para no escribir el nombre completo de la clase a la que hacemos referencia como: *com.compunauta.aprendiendojava.HolaMundo*
class	Inicia la declaración de una Clase (tipo de datos).
new	Instancia un objeto, crea un objeto partiendo de una clase en particular.

Tabla 4: Palabras clave - Permisos y Declaraciones

Bucles y tomas de decisión

Se utilizan para el control de bucles y tomas de decisión en un programa, se utilizan paréntesis () para encerrar las condiciones, y llaves {} para los bloques de datos

Palabra clave	*Significado*

if (condicion){
....
} **else** {
....
}
	Ejecuta el bloque separado por {} siempre y cuando se cumpla la condición, acompañada a esta estructura existe else (sinó) que es para el caso contrario a la condición, sin tener que evaluar la condición negada.

for (inicial;condición;repetición){
....
}
	Se ejecuta la sentencia inicial, y si se cumple la condición se ejecuta el código en el interior de la llaves, una vez que se termina se ejecuta la repetición y se vuelve a comparar la condición, mientras la condición sea cierta el bloque encerrado entre llaves se seguirá ejecutando.

while (condición){
....
}
	Mientras se cumpla la condición entre paréntesis, se ejecuta el código en el interior de {} de manera infinita hasta.

switch (varieblenumérica){
case num1:
.....
break;
case num2:
.....
break;
default:
}
	La sentencia switch es un tanto compleja y probabilísticamente poco usada ya que no puede usarse con objetos y las comparaciones son primitivas, en algunos trozos o segmentos de código es crucial para optimizar la velocidad de nuestro programa, de acuerdo al contenido de la variable numérica se ejecuta la sentencia case que coincida, cada bloque de código debe terminar con break, ya que en caso contrario se seguirá ejecutando la siguiente cláusula case, existe una opción default que es para cuando ninguno de los casos resulta ser el correcto.

do{ ... }**while**(condición);	El bloque dentro de la palabra clave do, se ejecutará al menos una vez y si la condición de while se cumple, se repetirá mientras sea cierta.
break	Interrumpe un bloque y se ejecuta la siguiente línea fuera de él.
continue	Interrumpe el resto del código del bloque y vuelve a empezar con la siguiente iteración.
return	Interrumpe el método entero y devuelve el tipo de datos de la derecha al punto donde se llamó a la función.

Tabla 5: Palabras Clave - Control de Flujos, tomas de decisión

Reservadas

Palabra clave	*Significado*
goto	Palabra reservada en el lenguaje de programación Java, no puede utilizarse pero el compilador en ciertas ocasiones es capaz de generar un comportamiento similar en las optimizaciones, de tal manera que los binarios si podrían estar utilizando este tipo de salto a una etiqueta.
const	Palabra reservada que no es utilizada por el momento en Java.

Tabla 6: Palabras clave – Reservadas

Excepciones, control de errores

En casi todos los lenguajes de programación existen mecanismos para actuar según sucedan errores de cualquier tipo, desde accesos al disco hasta división por cero, los cuales de no ser manipulados por nuestro programa tendrán el comportamiento por defecto de la máquina virtual de java, que es reportarlos como excepciones y terminar el programa.

Palabra clave	*Significado*
throws	Indica que tipo de excepciones lanzará este método en caso de errores, es útil para que los métodos que llaman a este que lanzará una excepción en caso de errores, se vean obligados a tratar esa situación.
throw	Transfiere el control de errores al manejador de excepciones.

try{
...
}**catch(tipo** excepcion){
...
}**finally**{
...
}
	Esta es la estructura de un manejador de excepciones o control de errores, try inicia el bloque de código que será manejado en caso de errores, la sentencia catch indica el tipo de excepción que se capturará, esta última puede repetirse para hacer cosas diferentes de acuerdo por ejemplo si el problema fue una división por cero o un error de acceso de disco. La sentencia finally se ejecutará de todas maneras al salir del código.
	Si una excepción no es capturada por el listado de cláusulas catch, entonces es probable que la JVM inicie el reporte y la salida de la instancia completa de la JVM, interrumpiendo todos los hilos de ejecución.

Tabla 7: Palabras clave - Control de errores

Secuencias de escape

Las secuencias de escape son aquellas combinaciones de caracteres que nos permiten insertar un caracter especial que no es posible tipear con el teclado estándar o que no es posible utilizar porque denota un trozo de texto, por ejemplo las comillas dobles en el ejemplo hola mundo que están dentro de la función *println*.

```
System.out.println("Hola, mi primer programa");
```

Si quisiéramos incluir las comillas dobles en la ejecución deberíamos incluirlas con una secuencia de escape, para ello se utiliza la barra invertida \ seguida de las comillas dobles \".

```
System.out.println("\"Hola, mi primer programa\"");
```

De esa manera el compilador entiende que el trozo de texto contendrá las comillas dobles, pero que pasaría si quisiéramos escribir el caracter \ que en algún sistema operativo distinto de Linux podría significar división de directorios o carpetas, bueno para esto usaríamos una \ seguida de otra \, es decir \\ para insertar el caracter \.

```
System.out.println("Hola, mi primer programa, está en c:\\");
```

Aquí hay una lista de las secuencias de escape más usadas y una breve explicación de su utilidad, ya que Java soporta Unicode, una representación extensa de lo que es un caracter que permite inclusión de varios idiomas.

Secuencia	*Significado*
\b	Retroceder un espacio (backspace)
\t	Tabulador Horizontal, sirve para alinear texto en la salida estándar de datos cuando es una terminal o consola de comandos.
\n	Salto de línea, es el caracter usado incluso en las viejas impresoras que necesitaban avanzar de línea para escribir en el próximo renglón.
\r	Retorno de carro, es el caracter que devuelve el cursor o el cabezal de la imprezora al inicio del renglón, Linux, usa \n para un nuevo renglón en los archivos de texto, Windows utiliza una combinación \n\r.
\”	Insertan las comillas dobles
\'	Insertan la coma simple.
\\	Insertan la misma barra invertida que usamos como escape.
\DDD	Las tres D son dígitos y representan un caracter en su versión octal.
\uDDDD	Las cuatro D son dígitos y representan un caracter unicode en hexadecimal.

Concatenación y conversiones a texto

La concatenación de texto se denomina a la unión de trozos o fragmentos de textos y comúnmente se utiliza la conversión de números o de objetos a su representación en texto, Java convierte automáticamente los tipos de datos básicos a texto antes de unirlos y llama por cada objeto al método especial toString() para realizar la conversión de objetos nuestros o de la biblioteca de la JVM.

Veamos un ejemplo:

```
1. public class SumaSimple {
2.   public static void main(String[] args) {
3.     int a=1;
4.     System.out.println("el valor de a="+a);
5.     a=a+10;
6.     System.out.println("ahora sumándole 10 es a="+a);
7.   }
8. }
```

En la línea 1, definimos el nombre de la clase, que es pública y se llama *SumaSimple*, en la segunda línea definimos el método estático *main* que es el que la JVM ejecutará para iniciar nuestros programas.

> ***Nota: El arreglo o vector args* se utiliza para recibir todas las órdenes de la JVM, los vectores en Java son objetos que tienen propiedades.**

En la línea 3, se define una variable de tipo entero llamada “a” que inicialmente guardará un 1 positivo, la 4ta línea ejecuta el método *println(String line)* que pertenece al objeto out que a su vez es propiedad del objeto System, tener un objeto de salida, se imprimirá en pantalla el trozo de texto *“el valor de a=”* seguido del valor (el contenido de a).

En la línea 5, al contenido de la variable a, le agregamos el resultado de sumar a+10, eso no tiene que dar un total de 11 que se verá reflejado en la línea 6, cuando volvemos a llamar al método *println* del objeto out que pertenece al objeto System.

En las líneas 7 y 8 se cierran las llaves que iniciaron los bloques de código del método *main* o la declaración de la clase *SumaSimple*.

> **Nota: Sólo como comentario lo que en realidad el compilador está haciendo al llamar a println, es concatenar texto, para ello es necesario convertir el número en texto, internamente el compilador estará llamando a una clase que es parte de la biblioteca cuyo nombre es Integer y posee un método toString() que convertirá el número, a su vez, el trozo de texto pertenecerá a la propiedad de un objeto String el cual tiene un método concat() que es el encargado de producir un nuevo objeto igual a la suma de los dos trozos de texto.**

El ejemplo podría escribirse así:

```
1. public class SumaSimple2 {
2.   public static void main(String[] args) {
3.     int a=1;
4.      System.out.println("el valor de a=".concat(Integer.toString(a)));
5.      a=a+10;
6.      System.out.println("ahora sumándole 10 es
   a=".concat(Integer.toString(a)));
7.   }
8. }
```

Donde se ha reemplazado la concatenación del compilador por su equivalente, recordemos que no es necesario hacer esto, vemos que después de las comillas dobles encontramos un punto, eso es porque de esa manera se representa un objeto String, cualquier trozo de texto entre comillas dobles es un objeto String y tiene propiedades y el punto separa objetos de otros objetos, propiedades o métodos.

> ***Nota: Las líneas que no empiezan con número representan la continuación de la anterior que por cuestiones de espacio no entra en la misma línea, esa representación es la misma que en muchos editores de código, al guardarse estos archivos se guardarán como una única línea, sin importar cuantos renglones sean necesarios para ver todo su contenido.***

En ambos casos la salida por pantalla será la misma y la siguiente:

```
gus@gusgus ~ $ cd ejemplos
gus@gusgus ~/ejemplos $ javac SumaSimple.java
gus@gusgus ~/ejemplos $ java SumaSimple
el valor de a=1
ahora sumándole 10 es a=11
gus@gusgus ~/ejemplos $ javac SumaSimple2.java
gus@gusgus ~/ejemplos $ java SumaSimple2
el valor de a=1
ahora sumándole 10 es a=11
gus@gusgus ~/ejemplos $
```

Salida por pantalla y entrada por teclado

Para los siguientes ejemplos ya tenemos casi todo discutido, excepto que por cuestiones educativas sería interesante que el programa no solo contenga los valores que interactuarán almacenados, sino que también sea posible preguntar al usuario.

System.out

El objeto System es parte de la biblioteca de Java y es instanciado o fabricado al iniciar la JVM mucho antes que se comience a ejecutar nuestro programa. Este almacena 3 objetos, son *out, in* y *err*.

El objeto *out* es del tipo o clase *Printstream* que tiene las siguientes propiedades y métodos importantes, no listaremos todos:

Resumen de Métodos	
boolean	**checkError**() Envía todo el buffer y devuelve verdadero si hay error o falso.
void	**close**() Cierra el flujo de datos
void	**flush**() Envía todo el buffer.
void	**print**(boolean b) Imprime una variable booleana
void	**print**(char c) Imprime un caracter.
void	**print**(char[] s) Imprime un arreglo de caracteres.
void	**print**(double d) Imprime un numero de tipo double.
void	**print**(float f) Imprime un número de punto flotante.
void	**print**(int i) Imprime un entero.
void	**print**(long l) Imprime un entero largo.
void	**print**(Object obj) Imprime un objeto, invocando su función toString()
void	**print**(String s) Imprime un objeto de tipo String
void	**println**() Imprime una separador de nueva línea.
void	**println**(boolean x) Imprime un valor booleano y termina la línea.
void	**println**(char x) Imprime un caracter y termina la línea.
void	**println**(char[] x) Imprime un arreglo de caracteres y termina la línea.
void	**println**(double x) Imprime un número de precisión doble y termina la línea.
void	**println**(float x) Imprime un número de punto flotante y termina la línea.
void	**println**(int x) Imprime un entero y termina la línea.

void	**println**(long x) Imprime un entero largo y termina la línea.
void	**println**(Object x) Imprime un objeto invocando su método toString() y termina la línea.
void	**println**(String x) Imprime un trozo de texto y termina la línea.

Tabla 8: Resumen de métodos importantes para out.

System.in

El objeto *in* que es una propiedad de *System* es de la clase o tipo *InputStream*, que también es parte de la biblioteca de Java. Aquí vemos los métodos que nos interesan.

Resumen de Métodos	
int	**available**() Devuelve la cantidad de bytes que se pueden leer (o pasar por alto) desde esta entrada sin bloquear la próxima llamada a lectura.
void	**close**() Cierra esta entrada de datos y libera todos los recursos asociados.
abstract int	**read**() Lee el próximo byte de datos desde la entrada, espera por los datos.
int	**read**(byte[] b) Lee de la entrada los bytes que llenan el arreglo b, devuelve la cantidad de bytes que se almacenaron.
int	**read**(byte[] b, int off, int len) Lee hasta len bytes de datos adentro del arreglo de bytes b empezando en off.
long	**skip**(long n) Salta y destrulle los n caracteres de datos.

Tabla 9: Resumen de métodos importantes del tipo de objetos in.

System.err

Este objeto es del mismo tipo que *out* (Printstream) y tiene las mismas propiedades, en los sistemas operativos derivados de Unix como ser Linux, existe mucho la diferencia entre salida estándar de datos (System.out) y la salida estándar de errores (System.err), por ejemplo al imprimir todos los carteles de errores en un tipo de salida no afectaríamos a los resultados, un ejemplo sería el siguiente:

```
public class SumaSimple3 {
  public static void main(String[] args) {
   int a=1;
    System.err.println("el valor de a="+a);
    System.out.println(a);
    a=a+10;
    System.err.println("ahora sumándole 10 es a="+a);
    System.out.println(a);
 }
}
```

Si este código lo compilamos al ejecutarlo podemos separar los comentarios de los valores de a.

```
gus@gusgus ~ $ cd ejemplos
gus@gusgus ~/ejemplos $ javac SumaSimple3.java
gus@gusgus ~/ejemplos $ java SumaSimple3
el valor de a=1
1
ahora sumándole 10 es a=11
11
gus@gusgus ~/ejemplos $ java SumaSimple3 > resultados.txt
el valor de a=1
ahora sumándole 10 es a=11
gus@gusgus ~/ejemplos $ java SumaSimple3 2> errores.txt
1
11
gus@gusgus ~/ejemplos $
```

De esa manera generamos y almacenamos por separado los errores y los resultados.

Nota: Es muy buena práctica acostumbrarse a separar los datos de errores a cada flojo de datos err y out, ya que si el flujo fuera un protocolo de comunicación mezclaríamos los datos con los errores.

System.exit(int cod);

El método estático exit(cod) de la clase System interrumpe la ejecución total del programa y devuelve el control al sistema operativo, es decir la JVM termina por completo liberando los recursos. Un código de salida igual a cero es considerado como una ejecución satisfactoria del programa, un código diferente a este, se considera un error y sirve para que el sistema operativo u otro programa que llame al nuestro sepa de alguna manera que ocurrió.

Leer líneas de la entrada estándar.

Para leer líneas de la entrada estándar podemos utilizar una combinación de objetos de la biblioteca que nos permitirán simplemente esperar a que el usuario ingrese texto y presione enter, o podemos codificar la lectura secuencial de caracteres uno en uno hasta que se detecte un salto de línea, tengamos en cuenta que java es multiplataforma, y si el sistema operativo no es Linux, es probable que se introduzca más de un caracter para el salto de línea lo cual incrementa la complejidad del problema.

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. public class LeerRenglones {
4.   public static void main(String[] args) {
5.     BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
6.     System.out.println("Hola, ingresa tu nombre");
7.     String nombre;
8.     try {nombre = br.readLine();}
9.     catch (IOException ex) {ex.printStackTrace();System.exit(-1);}
10.    System.out.println("Hola, "+nombre+" ten un buen día");
11.    System.exit(0);}
12.}
```

Nota: Queda como ejercicio para el lector ver los métodos y propiedades de la documentación de la api proporcionada por Sun Microsystems (java.sun.com) para las clases BufferedReader, InputStreamReader y Exception.

Crear Objetos (de la biblioteca de Java)

En java para crear objetos utilizamos una palabra clave, **new**, esta misma crea un objeto del tipo indicado a su derecha, repasemos del ejemplo anterior la siguiente línea:

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

La clase BufferedReader, pertenece a la biblioteca de Java "java.io" la cual fue importada antes de la declaración de la clase *LeerRenglones*, InputStreamReader, pertenece a la misma biblioteca.

> ***Nota: En este punto, estamos utilizando y creando objetos pertenecientes a la biblioteca, como podemos ver la declaración de nuestra clase LeerRenglones no tiene propiedades ni métodos dinámicos que puedan llegar a ser métodos de algún objeto, sólo el método estático main que es el que nos permite iniciar la ejecución de código. También podríamos haber usado DataInputStream.***

Veamos que significa cada parte de esta línea, el concepto de *"BufferedReader br="* es una asignación y una declaración simultánea, como vimos antes el signo *"="* asigna a la variable u nombre de objeto de la izquierda el contenido de la derecha. Eso quiere decir que *"br"* será una variable que apuntará a un objeto del tipo *BufferedReader*, *br* no será un objeto, será el nombre con el que accederemos al objeto en memoria ram, si en la siguiente línea repitiéramos otra asignación a *br,* es decir "*br=...."* sin la declaración porque ya se sabe que es del tipo *BufferedReader*, entonces la variable *br* apuntará a otro objeto del tipo *BufferedReader*, el anterior sigue existiendo en memoria, pero será eliminado en la próxima ejecución del recolector de basura de la JVM.

Cada vez que se crea un objeto nuevo el nombre de la clase se utiliza como el de una función, en realidad esta función especial es el método llamado constructor de la clase que es el que fabricará e inicializará de acuerdo a las variables que recibe como argumentos uno nuevo del tipo de su clase. Esto lo veremos más adelante, por el momento sólo haremos mención de que el objeto *BufferedReader*, necesita otro objeto que no es del tipo *InputStream* como lo es el objeto *System.in*, En cambio necesita uno del tipo *InputStreamReader*, que es el que es capaz de transformar un objeto *InputStream,* en *InputStreamReader* que es requerido para que el constructor de *BufferedReader* fabrique un objeto *BufferedReader*, a partir de uno *InputStreamReader.*

> ***Nota: Todo este procedimiento tiene el único fin de tener una variable llamada br que apunta a un objeto de tipo BufferedReader que es capaz de leer líneas desde la entrada, como lo construimos con la entrada del teclado, nos leerá los renglones que pretendíamos sin codificar la entrada del salto de línea.***

El bloque de control de errores, try{}catch(){}

Como podemos ver en este ejemplo utilizamos la estructura de control de errores que es obligatoria para esa conversión que hicimos para poder leer líneas con el método readLine(), la estructura try{} encierra entre las llaves el código que puede producir una excepción (un error grave) que debe ser manejado por el bloque catch(Exception ex){}, en este caso solo capturamos la excepción que puede surgir y cualquier otra que no sea IOException no será capturada.

```
try {nombre = br.readLine();}
catch (IOException ex) {ex.printStackTrace();}
```

Catch captura un tipo de error dentro del bloque try{}, y el bloque encerrado entre llaves ejecutará código que debería manejar la excepción, en este caso ignoramos todo tipo de acción que podríamos llevar a cabo y usamos la variable *ex* del tipo *IOException* para imprimir en pantalla los errores con el método *printStackTrace* de dicho objeto.

Ejercicios

En esta y todas las secciones de ejercicios veremos la solución a uno o dos, y el resto quedarán para ejercitación del lector, en el material multimedia que se adjunta con este libro tal vez se encuentren otras soluciones a los demás problemas.

2.1. Entrada de Datos y conversiones. [if, try, catch]

Preguntar el nombre del usuario y su edad, mostrar cuantos años tendría en una década más y clasificar según su edad en A[0-25], B[26-50], C[51-...]. La salida por pantalla debería ser algo como la siguiente:

```
Nombre:?
Gustavo
Edad:?29
Usuario Gustavo de Categoría B, en una década tendrá 39 años.
```

Nota: Al preguntar por el nombre el cursor debe quedar debajo y al preguntar por la edad junto, utilizar la estructura try{ }catch(){ } y los bloques if(){ }. También buscar en la documentación de la API los métodos parseInt( ) de la clase Integer.

Solución:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2006.01.01
10. */
11.
12.public class Cap2Ej1 {
13.  public static void main(String[] args) {
14.    //Definimos el objeto br para leer líneas de la entrada
15.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
16.    //Definimos variables nombre, edad y categoría dándole valores por
   defecto
17.    String nombre="";
18.    int edad=0;
19.    char categoria='A';
20.    //Iniciamos el bloque que podrá producir errores
21.    try {
22.      System.out.println("Nombre:?");
23.      nombre = br.readLine();
24.      System.out.print("Edad:?");
25.      edad=Integer.parseInt(br.readLine());}
26.  //capturamos cualquier excepción que se pueda producir y la reportamos
27.   catch (Exception ex) {ex.printStackTrace(System.err);System.exit(-1);}
28.   //Como por defecto la categoría es A, revisamos si aumentamos a B o C
29.    if(edad>25){categoria='B';}
30.    if(edad>50){categoria='C';}
31.    //Imprimimos en pantalla la respuesta solicitada
32.    edad+=10;
33.    System.out.println("El usuario "+nombre+" de categoría "+categoria+" en
   una década tendrá "+edad+ " años");
34.    System.exit(0);
35.  }//final de main
```

```
36.}//final de la clase
```

Comentarios:

Capturamos *Exception*, porque pueden producirse dos tipos de errores, uno el de *IOException,* como vimos en el ejemplo de lectura de renglones de este capítulo y el otro al ingresar texto en vez de un número. Utilizamos System.exit(-1); para salir del programa inesperadamente. Es conveniente que el alumno reemplace la doble creación de objetos por la de *DataInputStream*, que para nuestro caso es el mismo comportamiento, pero tengamos en cuenta que la función *readLine()* de *DataInputStream* no está recomendada por los nuevos kits de desarrollo solo existe por compatibilidad.

2.2 *NumberFormatException while(){}*

Basados en el enunciado del ejemplo anterior, capturar correctamente los errores de entrada de datos respecto de los de conversión de texto en números. Imprimir en pantalla con *printStackTrace*, para el caso de entrada, y avisar al usuario que la edad no fue ingresada correctamente.

Idea: Utilizar un bloque while para forzar al usuario a ingresar denuevo su nombre y edad si se detecta un error.

2.3 Mezcla de bucles do{} while(); y for(;;){}

Pedir por teclado el nombre al usuario, y a continuación solicitar 10 puntuaciones de supuestos exámenes, para promediarlos, la salida por pantalla debería ser algo así:

```
Nombre?César
Examen 1? 10
Examen 2? 7
Examen 4? 8.5
....
Examen 10? 9.3
César, tu promedio es de 8.93
```

Nota: Utilizar variables que permitan almacenar decimales, revisar en la documentación las otras Clases que proveen métodos de conversión con decimales de texto a números. Y Buscar en la clase Math, los métodos estáticos que permitan redondear los decimales. Y por supuesto repetir el ingreso de los Exámenes que que hayan sido ingresados incorrectamente.

Solución:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2006.01.01
10. */
11.
12.public class Cap2Ej3 {
13.  public static void main(String[] args) {
14.    //Definimos el objeto br para leer líneas de la entrada
15.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
16.    //Definimos variables nombre, error y acumulador dándole valores por
   defecto
```

```
17.     String nombre="";
18.     float acumulador=0;
19.     boolean error;
20.     //Iniciamos el bloque que podrá producir errores, sólo para el nombre
21.     try {
22.       System.out.println("Nombre:?");
23.       nombre = br.readLine();}
24.    catch (Exception ex) {ex.printStackTrace(System.err);}
25.    //iniciamos una iteración del 0 al 9
26.    for(int i=0;i<10;i++){
27.    error=false;
28.    //iniciamos el bloque do{} while(); que se repetirá en caso de error
29.    do{
30.    error=false;
31.    //iniciamos el bloque try que podrá dar error de conversión numérica
32.    try{
33.      //ponemos i+1 entre (), caso contrario se concatenarán como texto
34.      System.out.print("Examen "+(i+1)+"? ");
35.      acumulador+=Float.parseFloat(br.readLine());
36.    }catch(NumberFormatException ex){System.out.println("Error, ingresar
   denuevo");error=true;}
37.    catch(IOException ex){ex.printStackTrace();System.exit(-1);}
38.    }while (error);
39.}
40. //Tenemos lista la suma parcial y calculamos su promedio.
41. acumulador/=10;
42. //Redondeamos el resultado a dos digitos.
43.acumulador=(float)Math.round(acumulador*100)/100;
44. System.out.println(nombre+", tu promedio es de: "+acumulador);
45. System.exit(0);
46.}
47.}
```

2.4 Switch Select

Hacer un programa que utilice la estructura switch() para mostrar una frase de acuerdo a un número, pedir por teclado un número del 1 al 10, exigir que no se pase de esos valores, capturar errores y repetir hasta que se introduzca el 0 que será que sale del programa. Las frases inventarlas.

Práctica Complementaria Resuelta (sin procedimientos, sin arreglos)

Esta práctica se añadió con el objeto de que el alumno cuente con problemas resueltso para practicar la sintaxis del lenguaje, están resueltos y no utilizan bucles ni arreglos (que los veremos en el próximo capítulo).

P.C.E1

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
```

```
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej1 {
16.  /**
17.   * Ejercicio1: Un programa que carga por teclado dos números y obtiene y
18.   *             muestra la suma de ambos
19.   */
20.
21.    public static void main(String[] args) {
22.      int numero1 = 0;
23.      int numero2 = 0;
24.      int resultado;
25.      BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
26.      System.out.print("Ingrese el primer numero: ");
27.      try {
28.        numero1 = Integer.parseInt(br.readLine());
29.      }
30.      catch (IOException e) {
31.        e.printStackTrace(System.err);
32.        System.out.println("el programa se debe finalizar");
33.        System.exit( -1);
34.      }
35.      catch (Exception e) {
36.        e.printStackTrace(System.err);
37.        System.out.println("Error imprevisto");
38.        System.exit( -1);
39.      }
40.      System.out.print("Ingrese el segundo numero: ");
41.      try {
42.        numero2 = Integer.parseInt(br.readLine());
43.      }
44.      catch (IOException e) {
45.        e.printStackTrace(System.err);
46.        System.out.println("el programa se debe finalizar");
47.        System.exit( -1);
48.      }
49.      catch (Exception e) {
50.        e.printStackTrace(System.err);
51.        System.out.println("Error imprevisto");
52.        System.exit( -1);
53.      }
54.      resultado = numero1 + numero2;
55.      System.out.print("El resultado es: " + resultado);
56.    }
57.  }
```

P.C.E2

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
```

```
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.
16.public class Comp0Ej2 {
17.  /**
18.   * Ejercicio2: Un programa que carga por teclado el nombre de una persona y le
19.   *             muestra un saludo
20.   */
21.
22.          public static void main (String[] args)
23.          {
24.                  String nombre=new String("");
25.                  BufferedReader br=new BufferedReader(new
   InputStreamReader(System.in));
26.                  System.out.print("Ingrese su nombre: ");
27.                  try
28.                  {
29.                          nombre=br.readLine();
30.                  }
31.                  catch(IOException e)
32.                  {
33.                          e.printStackTrace(System.err);
34.                          System.out.println("el programa se debe
   finalizar");
35.                          System.exit(-1);
36.                  }
37.                  System.out.println("Hola "+nombre);
38.          }
39.  }
```

P.C.E3

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej3 {
16.  /**
17.   * Ejercicio3: Dado el valor de los tres lados de un triángulo, calcular el
18.   *             perímetro
19.   */
20.
21.  public static void main(String[] args) {
22.    int lado1 = 0;
23.    int lado2 = 0;
24.    int lado3 = 0;
25.    int perimetro;
26.    BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
```

```
27.     System.out.print("Ingrese el primer lado del triangulo: ");
28.     try {
29.       lado1 = Integer.parseInt(br.readLine());
30.     }
31.     catch (IOException e) {
32.       e.printStackTrace(System.err);
33.       System.out.println("el programa se debe finalizar");
34.       System.exit( -1);
35.     }
36.     catch (Exception e) {
37.       e.printStackTrace(System.err);
38.       System.out.println("Error imprevisto");
39.       System.exit( -1);
40.     }
41.     System.out.print("Ingrese el segundo lado del triangulo: ");
42.     try {
43.       lado2 = Integer.parseInt(br.readLine());
44.     }
45.     catch (IOException e) {
46.       e.printStackTrace(System.err);
47.       System.out.println("el programa se debe finalizar");
48.       System.exit( -1);
49.     }
50.     catch (Exception e) {
51.       e.printStackTrace(System.err);
52.       System.out.println("Error imprevisto");
53.       System.exit( -1);
54.     }
55.     System.out.print("Ingrese el tercer lado del triangulo: ");
56.     try {
57.       lado3 = Integer.parseInt(br.readLine());
58.     }
59.     catch (IOException e) {
60.       e.printStackTrace(System.err);
61.       System.out.println("el programa se debe finalizar");
62.       System.exit( -1);
63.     }
64.     catch (Exception e) {
65.       e.printStackTrace(System.err);
66.       System.out.println("Error imprevisto");
67.       System.exit( -1);
68.     }
69.     perimetro = lado1 + lado2 + lado3;
70.     System.out.println("El Perimetro del triangulo es: " + perimetro);
71.   }
72.}
```

P.C.E4

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6.
7. /**
8.  * <p>Título: Aprendiendo Java</p>
9.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
10. * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
11. * <p>Empresa: julioduarte@gmail.com</p>
```

```
12. * @author Julio César Duarte
13. * @version 2006.01.01
14. */
15.
16.
17.public class Comp0Ej4 {
18.  /**
19.   * Ejercicio4: Se conocen dos números. Determinar y mostrar el mayor
20.   */
21.  public static void main(String[] args) {
22.    int numero1 = 0;
23.    int numero2 = 0;
24.    BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
25.    System.out.print("Ingrese el primer numero: ");
26.    try {
27.      numero1 = Integer.parseInt(br.readLine());
28.    }
29.    catch (IOException e) {
30.      e.printStackTrace(System.err);
31.      System.out.println("el programa se debe finalizar");
32.      System.exit( -1);
33.    }
34.    catch (Exception e) {
35.      e.printStackTrace(System.err);
36.      System.out.println("Error imprevisto");
37.      System.exit( -1);
38.    }
39.    System.out.print("Ingrese el segundo numero: ");
40.    try {
41.      numero2 = Integer.parseInt(br.readLine());
42.    }
43.    catch (IOException e) {
44.      e.printStackTrace(System.err);
45.      System.out.println("el programa se debe finalizar");
46.      System.exit( -1);
47.    }
48.    catch (Exception e) {
49.      e.printStackTrace(System.err);
50.      System.out.println("Error imprevisto");
51.      System.exit( -1);
52.    }
53.    if (numero1 > numero2) {
54.      System.out.println("El numero mayor es: " + numero1);
55.    }
56.    else {
57.      if (numero1 < numero2) {
58.        System.out.println("El numero mayor es: " + numero2);
59.      }
60.      else {
61.        System.out.println("Los dos numeros son iguales: " + numero1);
62.      }
63.    }
64.  }
65.}
```

P.C.E5

```
1. package com.compunauta.aprendiendojava.ex;
```

```
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej5 {
16.  /**
17.   * Ejercicio5: Se conocen dos números distintos.
18.   * Determinar si el primero de ellos es el mayor
19.   */
20.    public static void main(String[] args) {
21.      int numero1 = 0;
22.      int numero2 = 0;
23.      BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
24.      System.out.print("Ingrese el primer numero: ");
25.      try {
26.        numero1 = Integer.parseInt(br.readLine());
27.      }
28.      catch (IOException e) {
29.        e.printStackTrace(System.err);
30.        System.out.println("el programa se debe finalizar");
31.        System.exit( -1);
32.      }
33.      catch (Exception e) {
34.        e.printStackTrace(System.err);
35.        System.out.println("Error imprevisto");
36.        System.exit( -1);
37.      }
38.      System.out.print("Ingrese el segundo numero distinto del primero: ");
39.      try {
40.        numero2 = Integer.parseInt(br.readLine());
41.      }
42.      catch (IOException e) {
43.        e.printStackTrace(System.err);
44.        System.out.println("el programa se debe finalizar");
45.        System.exit( -1);
46.      }
47.      catch (Exception e) {
48.        e.printStackTrace(System.err);
49.        System.out.println("Error imprevisto");
50.        System.exit( -1);
51.      }
52.      if (numero1 > numero2) {
53.        System.out.println("Se confirma que el primer numero es mas
   grande");
54.      }
55.      else {
56.        System.out.println("El primer numero no resulta ser el mas grande");
57.      }
58.
59.    }
60.  }
```

P.C.E6

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej6 {
16.  /**
17.   * Ejercicio6: Se conocen dos números distintos. Calcular la superficie de un
18.   *             cuadrado, suponiendo como lado del mismo al mayor de los números
19.   *             dados y la superficie de n círculo suponiendo como radio del
20.   *             mismo al menor de los números dados.
21.   */
22.
23.  public static void main(String[] args) {
24.    int numero1 = 0;
25.    int numero2 = 0;
26.    int mayor = 0;
27.    int menor = 0;
28.    int cuadrado = 0;
29.    double circulo = 0;
30.    BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
31.    System.out.print("Ingrese el primer numero: ");
32.    try {
33.      numero1 = Integer.parseInt(br.readLine());
34.    }
35.    catch (IOException e) {
36.      e.printStackTrace(System.err);
37.      System.out.println("el programa se debe finalizar");
38.      System.exit( -1);
39.    }
40.    catch (Exception e) {
41.      e.printStackTrace(System.err);
42.      System.out.println("Error imprevisto");
43.      System.exit( -1);
44.    }
45.    System.out.print("Ingrese el segundo numero distinto del primero: ");
46.    try {
47.      numero2 = Integer.parseInt(br.readLine());
48.    }
49.    catch (IOException e) {
50.      e.printStackTrace(System.err);
51.      System.out.println("el programa se debe finalizar");
52.      System.exit( -1);
53.    }
54.    catch (Exception e) {
55.      e.printStackTrace(System.err);
56.      System.out.println("Error imprevisto");
57.      System.exit( -1);
58.    }
59.    if (numero1 > numero2) {
60.      mayor = numero1;
```

```
61.       menor = numero2;
62.     }
63.     else {
64.       mayor = numero2;
65.       menor = numero1;
66.     }
67.     cuadrado = mayor * mayor;
68.     circulo = Math.PI * menor * menor;
69.     System.out.println("La supercie del cuadrado es: " + cuadrado);
70.     System.out.println("La supercie del circulo es: " + circulo);
71.   }
72.}
```

P.C.E7

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej7 {
16.   /**
17.    * Ejercicio7: Se conocen tres números distintos. Determinar el menor de ellos y
18.    *             cacular el cuadrado y el cubo del mismo
19.    */
20.
21.   public static void main(String[] args) {
22.     int numero1 = 0;
23.     int numero2 = 0;
24.     int numero3 = 0;
25.     int menor;
26.     BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
27.     System.out.print("Ingrese el primer numero: ");
28.     try {
29.       numero1 = Integer.parseInt(br.readLine());
30.     }
31.     catch (IOException e) {
32.       e.printStackTrace(System.err);
33.       System.out.println("el programa se debe finalizar");
34.       System.exit( -1);
35.     }
36.     catch (Exception e) {
37.       e.printStackTrace(System.err);
38.       System.out.println("Error imprevisto");
39.       System.exit( -1);
40.     }
41.     System.out.print("Ingrese el segundo numero: ");
42.     try {
43.       numero2 = Integer.parseInt(br.readLine());
44.     }
```

```
45.     catch (IOException e) {
46.       e.printStackTrace(System.err);
47.       System.out.println("el programa se debe finalizar");
48.       System.exit( -1);
49.     }
50.     catch (Exception e) {
51.       e.printStackTrace(System.err);
52.       System.out.println("Error imprevisto");
53.       System.exit( -1);
54.     }
55.     System.out.print("Ingrese el tercer numero: ");
56.     try {
57.       numero3 = Integer.parseInt(br.readLine());
58.     }
59.     catch (IOException e) {
60.       e.printStackTrace(System.err);
61.       System.out.println("el programa se debe finalizar");
62.       System.exit( -1);
63.     }
64.     catch (Exception e) {
65.       e.printStackTrace(System.err);
66.       System.out.println("Error imprevisto");
67.       System.exit( -1);
68.     }
69.     if (numero1 < numero2) {
70.       menor = numero1;
71.     }
72.     else {
73.       menor = numero2;
74.     }
75.     if (menor > numero3) {
76.       menor = numero3;
77.     }
78.     System.out.println("El numero menor es: " + menor);
79.     System.out.println("El cuadrado es: " + menor * menor);
80.     System.out.println("El cubo es: " + menor * menor * menor);
81.   }
82.}
```

P.C.E8

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej8 {
16.  /**
17.   * Ejercicio8: Se ingresan por teclado las notas obtenidas por tres alumnos en
18.   *             un parcial de ciertas materia. Se desea saber cuáles de estos
19.   *             alumnos resultaron aplazados, y además se pide determinar cuál
```

```
20.    *             fue la mayor nota, y cuál fue el alumno que la obtuvo.
21.    */
22.   public static void main(String[] args) {
23.     int nota1 = 0;
24.     int nota2 = 0;
25.     int nota3 = 0;
26.     int mejor;
27.     int alumno;
28.     BufferedReader br = new BufferedReader(new
    InputStreamReader(System.in));
29.     System.out.print("Ingrese la nota del primer alumno: ");
30.     try {
31.       nota1 = Integer.parseInt(br.readLine());
32.     }
33.     catch (IOException e) {
34.       e.printStackTrace(System.err);
35.       System.out.println("el programa se debe finalizar");
36.       System.exit( -1);
37.     }
38.     catch (Exception e) {
39.       e.printStackTrace(System.err);
40.       System.out.println("Error imprevisto");
41.       System.exit( -1);
42.     }
43.     System.out.print("Ingrese la nota del segundo alumno: ");
44.     try {
45.       nota2 = Integer.parseInt(br.readLine());
46.     }
47.     catch (IOException e) {
48.       e.printStackTrace(System.err);
49.       System.out.println("el programa se debe finalizar");
50.       System.exit( -1);
51.     }
52.     catch (Exception e) {
53.       e.printStackTrace(System.err);
54.       System.out.println("Error imprevisto");
55.       System.exit( -1);
56.     }
57.     System.out.print("Ingrese la nota del tercer alumno: ");
58.     try {
59.       nota3 = Integer.parseInt(br.readLine());
60.     }
61.     catch (IOException e) {
62.       e.printStackTrace(System.err);
63.       System.out.println("el programa se debe finalizar");
64.       System.exit( -1);
65.     }
66.     catch (Exception e) {
67.       e.printStackTrace(System.err);
68.       System.out.println("Error imprevisto");
69.       System.exit( -1);
70.     }
71.     System.out.println("Alumno Aplazados: ");
72.     if (nota1 < 4) {
73.       System.out.println("-Primer alumno aplazado");
74.     }
75.     if (nota2 < 4) {
76.       System.out.println("-Segundo alumno aplazado");
77.     }
78.     if (nota3 < 4) {
79.       System.out.println("-Tercer alumno aplazado");
```

```
80.     }
81.     System.out.println("Alumno que obtuvo la mejor nota: ");
82.     if (nota1 > nota2) {
83.       mejor = nota1;
84.       alumno = 1;
85.     }
86.     else {
87.       mejor = nota2;
88.       alumno = 2;
89.     }
90.     if (mejor < nota3) {
91.       mejor = nota3;
92.       alumno = 3;
93.     }
94.     System.out.println("El alumno" + alumno + " fue quien obtuvo un: " +
   mejor);
95.   }
96.}
```

P.C.E9

```
1. package com.compunauta.aprendiendojava.ex;
2. import java.io.BufferedReader;
3. import java.io.InputStreamReader;
4. import java.io.IOException;
5.
6. /**
7.  * <p>Título: Aprendiendo Java</p>
8.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
9.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
10. * <p>Empresa: julioduarte@gmail.com</p>
11. * @author Julio César Duarte
12. * @version 2006.01.01
13. */
14.
15.public class Comp0Ej9 {
16.  /**
17.   * Ejercicio9: Un comerciante tiene la venta de 4 tipos de productos principales
18.   *             Conociendo la cantidad vendida de cada artículo, y el precio
19.   *             unitario de cada artículo, hacer un programa que determine cuál
20.   *             fue el producto que realizó el mayor aporte en los ingresos.
21.   *             Calcular además, el porcentaje que dicho aporte significa en el
22.   *             ingreso absoluto por los cuatro artículos sumados.
23.   */
24.
25.  public static void main(String[] args) {
26.    String buf = new String("");
27.    float precio1 = 0;
28.    int cantidad1 = 0;
29.    float aporte1 = 0;
30.    float precio2 = 0;
31.    int cantidad2 = 0;
32.    float aporte2 = 0;
33.    float precio3 = 0;
34.    int cantidad3 = 0;
35.    float aporte3 = 0;
36.    float precio4 = 0;
37.    int cantidad4 = 0;
38.    float aporte4 = 0;
39.    float aportetotal = 0;
40.    float mayor = 0;
```

```
41.     int id = 0;
42.     float porcentaje = 0;
43.     BufferedReader br = new BufferedReader(new
   InputStreamReader(System.in));
44.     System.out.print("Ingrese el precio producto1: ");
45.     try {
46.       //precio1=Double.valueOf(br.readLine());
47.       precio1 = Float.valueOf(br.readLine()).floatValue();
48.     }
49.     catch (IOException e) {
50.       e.printStackTrace(System.err);
51.       System.out.println("el programa se debe finalizar");
52.       System.exit( -1);
53.     }
54.     catch (Exception e) {
55.       e.printStackTrace(System.err);
56.       System.out.println("Error imprevisto");
57.       System.exit( -1);
58.     }
59.     System.out.print("Ingrese el precio producto2: ");
60.     try {
61.       precio2 = Float.valueOf(br.readLine()).floatValue();
62.     }
63.     catch (IOException e) {
64.       e.printStackTrace(System.err);
65.       System.out.println("el programa se debe finalizar");
66.       System.exit( -1);
67.     }
68.     catch (Exception e) {
69.       e.printStackTrace(System.err);
70.       System.out.println("Error imprevisto");
71.       System.exit( -1);
72.     }
73.     System.out.print("Ingrese el precio producto3: ");
74.     try {
75.       precio3 = Float.valueOf(br.readLine()).floatValue();
76.     }
77.     catch (IOException e) {
78.       e.printStackTrace(System.err);
79.       System.out.println("el programa se debe finalizar");
80.       System.exit( -1);
81.     }
82.     catch (Exception e) {
83.       e.printStackTrace(System.err);
84.       System.out.println("Error imprevisto");
85.       System.exit( -1);
86.     }
87.     System.out.print("Ingrese el precio producto4: ");
88.     try {
89.       precio4 = Float.valueOf(br.readLine()).floatValue();
90.     }
91.     catch (IOException e) {
92.       e.printStackTrace(System.err);
93.       System.out.println("el programa se debe finalizar");
94.       System.exit( -1);
95.     }
96.     catch (Exception e) {
97.       e.printStackTrace(System.err);
98.       System.out.println("Error imprevisto");
99.       System.exit( -1);
100.     }
```

```
101.    System.out.print("Ingrese la cantida vendida del producto1: ");
102.    try {
103.      cantidad1 = Integer.parseInt(br.readLine());
104.    }
105.    catch (IOException e) {
106.      e.printStackTrace(System.err);
107.      System.out.println("el programa se debe finalizar");
108.      System.exit( -1);
109.    }
110.    catch (Exception e) {
111.      e.printStackTrace(System.err);
112.      System.out.println("Error imprevisto");
113.      System.exit( -1);
114.    }
115.    System.out.print("Ingrese la cantida vendida del producto2: ");
116.    try {
117.      cantidad2 = Integer.parseInt(br.readLine());
118.    }
119.    catch (IOException e) {
120.      e.printStackTrace(System.err);
121.      System.out.println("el programa se debe finalizar");
122.      System.exit( -1);
123.    }
124.    catch (Exception e) {
125.      e.printStackTrace(System.err);
126.      System.out.println("Error imprevisto");
127.      System.exit( -1);
128.    }
129.    System.out.print("Ingrese la cantida vendida del producto3: ");
130.    try {
131.      cantidad3 = Integer.parseInt(br.readLine());
132.    }
133.    catch (IOException e) {
134.      e.printStackTrace(System.err);
135.      System.out.println("el programa se debe finalizar");
136.      System.exit( -1);
137.    }
138.    catch (Exception e) {
139.      e.printStackTrace(System.err);
140.      System.out.println("Error imprevisto");
141.      System.exit( -1);
142.    }
143.    System.out.print("Ingrese la cantida vendida del producto4: ");
144.    try {
145.      cantidad4 = Integer.parseInt(br.readLine());
146.    }
147.    catch (IOException e) {
148.      e.printStackTrace(System.err);
149.      System.out.println("el programa se debe finalizar");
150.      System.exit( -1);
151.    }
152.    catch (Exception e) {
153.      e.printStackTrace(System.err);
154.      System.out.println("Error imprevisto");
155.      System.exit( -1);
156.    }
157.    aporte1 = precio1 * cantidad1;
158.    aporte2 = precio2 * cantidad2;
159.    aporte3 = precio3 * cantidad3;
160.    aporte4 = precio4 * cantidad4;
161.    aportetotal = aporte1 + aporte2 + aporte3 + aporte4;
```

```
162.     if (aporte1 > aporte2) {
163.       mayor = aporte1;
164.       id = 1;
165.     }
166.     else {
167.       mayor = aporte2;
168.       id = 2;
169.     }
170.     if (mayor < aporte3) {
171.       mayor = aporte3;
172.       id = 3;
173.     }
174.     if (mayor < aporte4) {
175.       mayor = aporte4;
176.       id = 4;
177.     }
178.     porcentaje = (mayor / aportetotal) * 100;
179.     System.out.println("El producto" + id + " fue el que mas aporto con: "
    +
180.                        mayor);
181.     System.out.println("El porentaje de aporte sobre el total es de: " +
182.                        porcentaje + "%");
183.
184.   }
185.}
```

Práctica Complementaria (bucles sin arreglos)

III – Métodos estáticos y Mecanismos de Programación.

Este capítulo pretende proporcionar al usuario final las herramientas para solucionar problemas del tipo programación estructurada, de pilas, colas, vectores, árboles, etc. Que es necesario para formar correctamente al alumno en cualquier ambiente de programación, incluida la programación orientada a objetos, no pretendemos ir a la P.O.O todavía sino hasta más adelante, de esta manera este capítulo puede utilizarse para afianzar los conocimientos necesarios para un curso de física o matemáticas donde se desee aplicar a problemas en general.

La P.O.O. en este capítulo será completamente básica, y ni siquiera la mencionaremos en muchos caso hasta el final del capítulo a menos que sea necesario.

Métodos estáticos (funciones o procedimientos)

En el capitulo anterior no utilizamos nada de la programación estructurada, sólo seguimos un orden de ejecución secuencial, es decir paso a paso y no planteamos el problema de la reutilización de código.

Problema: necesitamos ingresar datos por teclado, hacer varios cálculos y volver a ingresar datos por teclado y hacer otros cálculos diferentes, y este mismo proceso, repetidas veces, como vimos en los ejemplos anteriores tendríamos que repetir el bloque try{}catch(){} varias veces, y si quisiéramos repetir un bloque de cálculos para ciertos datos... **¿Que pasaría si quiero cambiar esas órdenes las veces que se hayan repetido?**.

Entonces comenzamos con las funciones, o métodos estáticos en Java, se les denomina estáticos porque pertenecen a la Clase o tipo de datos y no es necesario crear un objeto para invocarlos o llamarlos, para nosotros en este momento nos servirán para solucionar el problema de repetición y mantenimiento de código.

Ejemplo 3.1:

Se requiere ingresar los datos de 3 valores y calcular su promedio, paso seguido, preguntar el nombre del usuario e imprimir de manera personalizada “Usuario, el promedio de tus tres valores es: XXX”, es un problema muy similar a los anteriores, pero utilizaremos una función llamada leer que fabricaremos para no repetir el código de lectura del teclado.

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2006.01.01
10. */
11.
12.public class MetodoLeer {
13.  public static void main(String[] args) {
14.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
15.    int acumulador=0;
16.    for(int i=0;i<3;i++){
17.      System.out.println("Ingrese el valor "+(i+1)+" de 3?");
18.    acumulador=acumulador+=Integer.parseInt(leer(br));
19.    }
20.    acumulador/=3;
```

```
21.     System.out.println("Ingrese su nombre?");
22.     String nombre=leer(br);
23.     System.out.println("Usuario: "+nombre+" tu promedio es:"+acumulador);
24.   }
25.
26.public static String leer(BufferedReader buff){
27.   String lee="";
28.try{lee=buff.readLine();}
29.catch(Exception ex){
30.   ex.printStackTrace(System.err);}
31.return lee;
32.}//final de la funcion leer
33.}//final de la clase
```

Como podemos ver desde la línea 26 a la 32 se define la función estática leer, que es pública y devolverá datos de tipo String, esta función para poder procesar la lectura necesita que le pasemos como argumento un objeto del tipo *BufferedReader* que es el encargado de leer las líneas, esta función tiene como objetivo que no se repita el código de control de errores y en el futuro podremos modificar esta función o fabricar otra para leer números que revise una correcta entrada de datos.

Bien, ahora veamos el código mejorado utilizando una función para leer texto y otra para los números enteros:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2006.01.01
10. */
11.
12.public class MetodoLeer {
13.  public static void main(String[] args) {
14.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
15.    int acumulador=0;
16.    for(int i=0;i<3;i++){
17.      System.out.println("Ingrese el valor "+(i+1)+" de 3?");
18.    acumulador=acumulador+=Integer.parseInt(leerTexto(br));
19.    }
20.    acumulador/=3;
21.    System.out.println("Ingrese su nombre?");
22.    String nombre=leerTexto(br);
23.    System.out.println("Usuario: "+nombre+" tu promedio es:"+acumulador);
24.  }
25.
26.public static String leerTexto(BufferedReader buff){
27.  String lee="";
28.try{lee=buff.readLine();}
29.catch(Exception ex){
30.  ex.printStackTrace(System.err);}
31.return lee;
32.}//final de la funcion leer
33.
34.public static int leerInt(BufferedReader buff){
35.  int lee=0;
36.  boolean error;
```

```
37.  do {
38.    error=false;
39.    try {lee = Integer.parseInt(buff.readLine());}
40.    catch (NumberFormatException ex) {
41.      System.out.println("Entrada erronea, repetir:?");
42.      error=true;}
43.    catch (Exception ex){ex.printStackTrace(System.err);}
44.  }  while (error);
45.return lee;
46.}//final de la funcion leer
47.
48.}
```

Arreglos (Arrays) o Vectores.

Los arreglos son como un vector, con varias componentes incluso en programación un arreglo puede representar una matriz de varias dimensiones. Por ejemplo si tenemos 10 frases que enseñaremos según un número del 1 al 10, podemos utilizar un vector de una sola dimensión, donde indicaremos con un subíndice a los elementos que este mismo almacene.

En Java un arreglo se representa por un objeto que tiene un límite de elementos al ser definido, o en alguna parte de nuestro programa le asignaremos un objeto arreglo de cierto tipo. Los elementos que pondremos en nuestro arreglo de datos deben estar definidos en cantidad, no en valor, si creemos que podemos necesitar más o menos 10+/-3 elementos asignaremos 13 por más que a veces usemos 7.

Los arreglos tienen una cantidad de elementos, pero el subíndice que usaremos para acceder al contenido de cada elemento irá del 0 a ELEMENTOS-1. Veamos un ejemplo:

Problema:

Se desea ingresar por teclado tres resultados de exámenes, e imprimir el 1º y último, utilizamos la función *leerInt()* que ejemplificamos anteriormente.

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2006.01.01
10. */
11.
12.public class Arreglos {
13.  public static void main(String[] args) {
14.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
15.    int[] examenes=new int[3];
16.    System.out.println("La longitud de datos de mi arreglo
   es:"+examenes.length);
17.    for (int i=0;i<examenes.length;i++){
18.    System.out.println("Ingrese el valor para el examen "+(i+1)+" de
   "+examenes.length);
19.    examenes[i]=leerInt(br);
20.    }
21.    System.out.println("El resultado del primer examen es:"+examenes[0]);
22.    System.out.println("El resultado del último examen
   es:"+examenes[examenes.length-1]);
23.  }
24.  public static int leerInt(BufferedReader buff){
```

```
25.  int lee=0;
26.  boolean error;
27.  do {
28.    error=false;
29.    try {lee = Integer.parseInt(buff.readLine());}
30.    catch (NumberFormatException ex) {
31.System.out.println("Entrada erronea, repetir:?");
32.      error=true;}
33.    catch (Exception ex){ex.printStackTrace(System.err);}
34.  }  while (error);
35.return lee;
36.}//final de la funcion leer
37.
38.}
```

La salida por pantalla será algo como esto:

```
La longitud de datos de mi arreglo es:3
Ingrese el valor para el examen 1 de 3
23
Ingrese el valor para el examen 2 de 3
21
Ingrese el valor para el examen 3 de 3
12
El resultado del primer examen es:23
El resultado del último examen es:12
```

La clase Math de procedimientos y constantes matemáticas

A continuación veremos la utilidad de la Clase de la biblioteca más útil, que provee algunas cuantas funciones matemáticas que usaremos para resolver nuestros problemas en este capítulo.

Resumen de Campos

static double	**E** El valor `double` que es el mas cercano a *e*, la base de los logaritmos naturales.
static double	**PI** El valor `double` que es más cercano a *pi*, la relación de una circunferencia con su diámetro.

Resumen de Métodos

static double	**abs**(double a) Devuelve el valor absoluto de un valor `double`.
static float	**abs**(float a) Devuelve el valor absoluto de un valor `float`.
static int	**abs**(int a) Devuelve el valor absoluto de un valor `int`.
static long	**abs**(long a) Devuelve el valor absoluto de un valor `long`.

static double	**acos**(double a) Devuelve el arcocoseno de un ángulo, en el rango de 0.0 hasta *pi*.
static double	**asin**(double a) Devuelve el arcoseno de un ángulo, en el rango de -*pi*/2 hasta *pi*/2.
static double	**atan**(double a) Devuelve el arcotangente de un ángulo, en el rango de -*pi*/2 hasta *pi*/2.
static double	**atan2**(double y, double x) Convierte coordenadas rectangulares (x, y) a polares (r, *theta*).
static double	**cbrt**(double a) Devuelve la raíz cuadrada de un valor double.
static double	**ceil**(double a) Devuleve el más pequeño (cercano al infinito negativo) valor double que es más grande o igual al argumento a y es igual a un entero matemático.
static double	**cos**(double a) Devuelve el coseno trigonométrico de un ángulo.
static double	**cosh**(double x) Devuelve el coseno hiperbólico de un valor value.
static double	**exp**(double a) Devuelve el valor *e* de Euler elevado a la potencia del valor double a.
static double	**expm1**(double x) Devuelve e^x -1.
static double	**floor**(double a) Devuelve el más largo (cercano al positivo infinito) valor double que es menor o igual al argumento a y es igual a un entero matemático.
static double	**hypot**(double x, double y) Devuelve sqrt(x^2 +y^2) sin el overflow o underflow intermedio.
static double	**IEEEremainder**(double f1, double f2) Computa la operación prescripta por el estándar IEEE 754 entre los dos argumentos f1 y f2.
static double	**log**(double a) Devuelve el logaritmo natural (base *e*) de un valor double.
static double	**log10**(double a) Devuelve el logaritmo en base 10 de un valor double.
static double	**log1p**(double x) devuelve e^{x+1}.
static double	**max**(double a, double b) Devuelve el más grande de los dos valores double a y b.
static float	**max**(float a, float b) Devuelve el más grande de los dos valores float a y b.
static int	**max**(int a, int b) Devuelve el más grande de los dos valores int a y b.
static long	**max**(long a, long b)

	Devuelve el más grande de los dos valores `long` a y b.
`static double`	**`min`**`(double a, double b)` Devuelve el más pequeño de los dos valores `double` a y b.
`static float`	**`min`**`(float a, float b)` Devuelve el más pequeño de los dos valores `float` a y b.
`static int`	**`min`**`(int a, int b)` Devuelve el más pequeño de los dos valores `int` a y b.
`static long`	**`min`**`(long a, long b)` Devuelve el más pequeño de los dos valores `long` a y b.
`static double`	**`pow`**`(double a, double b)` Devuelve el valor del argumento a elevado a la potencia de b: a^b .
`static double`	**`random`**`()` Devuelve un valor de tipo `double` con signo positivo,mayor o igual que cero y menor que uno `1.0`.
`static double`	**`rint`**`(double a)` Devuelve el valor `double` que es más cercano al valor a y es igual a un entero matemático.
`static long`	**`round`**`(double a)` Devuelve el valor `long` más cercano al argumento.
`static int`	**`round`**`(float a)` Devuelve el valor `int` más cercano al argumento.
`static double`	**`signum`**`(double d)` La función signo, cero si el argumento es cero, 1.0 si el argumento es mayor que cero y -1.0 si el argumento es menor que cero.
`static float`	**`signum`**`(float f)` La función signo, cero si el argumento es cero, 1.0 si el argumento es mayor que cero y -1.0 si el argumento es menor que cero.
`static double`	**`sin`**`(double a)` Devuelve el seno trigonométrico de un ángulo.
`static double`	**`sinh`**`(double x)` Devuelve el seno hiperbólico de un valor `double`.
`static double`	**`sqrt`**`(double a)` Devuelve la raíz cuadrada positiva redondeada de un valor `double`.
`static double`	**`tan`**`(double a)` Devuelve la tangente trigonométrica de un ángulo.
`static double`	**`tanh`**`(double x)` Devuelve la tangente hiperbólica de un valor `double`.
`static double`	**`toDegrees`**`(double angrad)` Convierte un ángulo medido en radianes al aproximado en grados..
`static double`	**`toRadians`**`(double angdeg)` Convierte un ángulo medido en grados al aproximado en radianes.
`static double`	**`ulp`**`(double d)`

	Ver definición en la documentación completa.
`static float`	**`ulp`**`(float f)` Ver definición en la documentación completa.

Tabla 10: La Clase Math - métodos y constantes

Buffering – Memoria temporal

Algunas veces es necesario procesar datos con cierta velocidad o cantidad conocida, la lectura de datos de 1 en 1 puede producir que en un cierto momento nuestro programa se quede sin hacer nada, o simplemente es más efectivo leer un archivo por bloques de datos grandes que de byte en byte (sobrecarga de llamadas a la misma función).

En otras ocasiones podemos estar recibiendo datos por la red y si nuestro programa es lento para procesarlos o necesita atención por parte de un usuario para decidir que hacer, puede suceder que lleguen más datos de los que se pueden procesar, en este caso podemos utilizar un segmento de memoria temporal para almacenar estos datos y que no se pierdan y así aprovechar el tiempo en el que no llegan datos.

El criterio para implementar estos tipos de memorias temporales (buffers de aquí en adelante) es variable, siempre se tomará en cuenta el promedio de datos que lleguen a nuestro programa o la cantidad de veces que es necesario llamar a una función que lee datos si lo hacemos con bloques más pequeños o más grandes, el valor óptimo siempre es empírico.

Para determinar este valor siempre tendremos en cuenta que bloques más grandes de datos consumen más memoria del sistema operativo, y que muchas veces para mover bloques grandes de datos se puede perder mucho tiempo, por otra parte si nuestro archivo del disco no dispone de datos para llenar esos bloques, estaríamos desperdiciando memoria si abriéramos muchos archivos más pequeños que el buffer, por otro lado si el buffer es muy pequeño, entonces el efecto es contrario, estaríamos llamando innumerables cantidades de veces a la función leer, por lo que produciríamos un desperdicio de recursos del procesador, enlenteceríamos el sistema operativo, entonces queda claro que no leeremos archivos en bloques de 1MB ni lo haremos byte por byte.

Usando arreglos para un buffer, colas de espera, pilas y listas.

Bien, pongamos el siguiente ejemplo, se quieren enviar datos de una máquina a otra en un sólo sentido, una telefonista debe llamar a ciertos teléfonos que llegan desde otro punto de una red y no puede ver el siguiente teléfono hasta que haya terminado su llamada, del otro lado una secretaria que revisa el correo con una frecuencia de 5 minutos envía los teléfonos del soporte técnico que deben ser atendidos por la telefonista, teniendo en cuenta que las llamadas suelen durar entre 10 a 30 minutos y que los correos que pueden llegar al mismo tiempo no son más de 6 y que existen lapsos de hasta 60 minutos sin recibir ni un sólo correo, y que no quedan llamadas pendientes en el transcurso del día.

Criterios:

1. Sobredimensionar es sobrecargar
2. Minimizar es sobrecargar
3. Si llenamos el buffer, no se podrán ingresar más dátos.
4. Supondremos que en el peor de los casos llegan 10 correos
5. Supondremos que en el peor de los casos duran 30 minutos
6. Supondremos que en el peor de los casos el lapso es de 20 minutos durante 4 períodos.

Los últimos tres criterios son suposiciones, que en la vida real son la telefonista y secretaria

quienes pueden proveer este tipo de estadística sobre los horarios más saturados, tenemos que tener en cuenta que el problema es humanamente soluble ya que no quedan pendientes sin tratar así que durante el día se resuelven todos los llamados: 20min x 4p = 80min => 1) se resuelven en 80 minutos 2 llamados y ¾ 2) se acumularon casi 40 tel – 3 tel = 37 tel

Entonces en el peor de los casos la acumulación máxima probable y el despacho de teléfonos más tardado producen un máximo de 37 teléfonos. Entonces nuestro ***buffer óptimo*** es de **37 elementos**. Como almacenar un teléfono no consume tanta cantidad de recursos de un sistema utilizaremos un buffer de 40 elementos.

> *Nota: el manejo de memoria RAM en el sistema para valores o tipos de datos de almacenamiento fijo, como son todos los tipos de datos básicos excluyendo "String", es de potencias de 2^n y a veces de múltiplos de estas potencias, siendo recomendable elegir siempre un valor cercano pero no igual porque si la JVM utiliza bloques de 64enteros como buffer interno si usáramos 65enteros estaríamos obligando a usar el espacio de 128enteros.*

Implementación del buffer tipo FIFO (Cola de espera, el primero es primero en salir)

El ejemplo anterior necesita conocimientos de ejecución paralela de dos procesos, uno para leer del socket (conexión de red) y otro para atender a la telefonista, eso no lo veremos todavía así que cambiaremos el enunciado para implementar esta cola de espera.

La telefonista recibe los teléfonos y tardará solo 1 minuto como máximo por llamada, es decir los números saldrán de a 1 por minuto, pudiendo descansar en el tiempo sobrante, si no hay números esperar, y si hay muchos guardarlos temporalmente.

Utilizaremos una conexión de red, donde la telefonista tendrá un programa en espera que escuchará en la red y la secretaria el que enviará los datos al establecer una conexión.

Telefonista:

```
gus@gusgus ~$ java com.compunauta.aprendiendojava.Cap3_sock_tel
Escuchando el puerto:4567
Esperando conexión...
Conectado... Esperando teléfonos
Secretaria llamando al tel:123
Secretaria llamando al tel:432
Ultima llamada, nos vamos... programa terminado
gus@gusgus ~$
```

Secretaria:

```
gus@gusgus ~$ java com.compunauta.aprendiendojava.Cap3_sock_sec
Intentando conectar con la telefonista
Nos conectamos con la telefonista:127.0.0.1:4567
Ingrese números -1 termina
123
432
-1
Programa terminado
gus@gusgus ~$
```

Codificación:

Como dijimos antes la implementación y codificación de este ejemplo necesitará de una conexión de red para poder comprender mejor el uso de este tipo de estructuras, no es necesario que dispongamos de una red para llevar a cabo este programa, ya que una PC puede actuar como una red de 1 sola

máquina, en este caso usaremos la dirección de red local (loopback) que existe en casi todos los sistemas operativos en los que corre Java. Esta dirección especial, permite conectarse por medio de los protocolos de red a la misma máquina en caso que estemos en un laboratorio de informática y dispongamos de la información de las direcciones ip de las otras PCs de la red sería interesante que se trabajara en grupos de dos, para poder apreciar la conexión remota.

Nota: Queda como tarea para el lector revisar los métodos y propiedades de la clase o tipo de datos Socket y ServerSocket.

Para establecer una conexión de red es necesario que alguien esté escuchando en un puerto, estos puertos son un concepto abstracto que podemos asimilarlo comparándolo con un puerto real donde llegan barcos, dichos barcos una vez que llegan son atendidos y siempre que haya quien atenderlos podrá entrar otro.

En este caso solo esperaremos una única conexión en el puerto, y cuando esta se establezca no esperaremos ninguna otra.

La clase ServerSocket nos permitirá fabricar un objeto que podrá esperar una conexión, y cuando esta llegue podemos abrirla por medio del objeto resultante Socket (conexión). El puerto de escucha en este ejemplo será: 4567, muchos firewalls y mecanismos de protección de red pueden bloquear el acceso a estos puertos, así que cualquier cosa preguntamos al administrador de la red, por otro lado no podemos usar un puerto más bajo inferior a los 1024 porque será necesario por lo general permisos especiales en el sistema operativo.

Telefonista:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. import java.net.*;
4. /**
5.  * <p>Título: Aprendiendo Java</p>
6.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
7.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
8.  * <p>Empresa: COMPUNAUTA</p>
9.  * @author Gustavo Guillermo Pérez
10. * @version 2006.01.01
11. */
12.
13.public class Cap3_sock_tel {
14.//Declaramos unas variables globales a este tipo de datos
15.public static int PORT=4567;
16.public static int BUFF_SIZE=40;
17.public static int TIMER_SLEEP=60*1000; //60sx1000ms
18.public static int buff_elem=0;
19.public static int[] buffer=new int[BUFF_SIZE];
20.
21.  public static void main(String[] args) {
22.    //Declaramos la variable socket (será un puntero a objeto)
23.    Socket skt=(Socket)null;
24.    //Declaramos vacío el servidor de sockets para inicializarlo
25.    ServerSocket Ss=(ServerSocket)null;
26.
27.    //Tratamos de escuchar el puerto definido por la variable PORT
28.    System.err.println("Escuchando el puerto:"+PORT);
29.    try {Ss = new ServerSocket(PORT);}
30.    catch (IOException ex) {
31.      System.out.println("El sistema no permite abrir el puerto");
32.      System.exit(-1);}
33.
34.    //Si no ocurrió error arriba entonces esperamos a la secretaria
```

```
35.     System.err.println("Esperando conexión...");
36.     try {skt = Ss.accept();}
37.     catch (IOException ex1) {
38.       ex1.printStackTrace(System.err);
39.       System.exit(-1);}
40.
41.     //Si no ocurrió error arriba la secretaria está lista para enviar
42.     System.err.println("Conectado... Esperando teléfonos");
43.     try {
44.       ObjectInputStream datos = new ObjectInputStream(skt.getInputStream());
45.       long timer=0;
46.       boolean timer_on=false;
47.       while (true){
48.         if((skt.isClosed() && (buff_elem<1)) || (buffer[0]==-1)){
49.         //Terminamos el programa si la secretaria terminó
50.         System.err.println("Ultima llamada, nos vamos... terminado");
51.         System.exit(0);
52.         }
53.         //si hay teléfonos los guardamos
54.         if(datos.available()>0){
55.         put_tel(datos.readInt());}
56.         if(timer_on){
57.           //si el timer funciona no hacer nada, si se pasó pararlo
58.         if ((timer+TIMER_SLEEP)<System.currentTimeMillis()){timer_on=false;}
59.         }else{
60.           //Si el timer está apagado, mostramos un tel si es que hay
61.           if (buff_elem>0){System.out.println("Secretaria llamando al
   tel:"+get_tel());
62.           //Encendemos el timer y guardamos la hora en que empezó
63.           timer_on=true;
64.           timer=System.currentTimeMillis();}
65.         }
66.         //Pausamos 100ms para no sobrecargar el procesador
67.         try {Thread.sleep(100);}
68.         catch (InterruptedException ex3) {}
69.       }//fin del bloque eterno
70.     }catch (IOException ex2) {
71.       ex2.printStackTrace(System.err);
72.       System.exit(-1);
73.     }
74.
75.   }//fin del método principal
76.
77.//Funciones o métodos auxiliares
78.public static void put_tel(int tel){
79.//Si se supera el espacio producir un error
80.if (BUFF_SIZE<(buff_elem+1)){
81.System.err.println("Buffer overrun: El buffer se llenó demasiado rápido");
82.System.exit(-1);}
83.//guardamos el tel y aumentamos en uno el contador
84.buffer[buff_elem++]=tel;
85.}
86.
87.public static int get_tel(){
88.  //almacenamos el primer teléfono
89.  int tel=buffer[0];
90.  //quitamos uno al contador de elementos
91.  buff_elem--;
92.  //recorremos los otros elementos
93.  for (int i=0;i<buff_elem;i++) buffer[i]=buffer[i+1];
94.  //devolvemos el primer teléfono
```

```
95.  return tel;
96.}
97.
98.}//final de la clase
```

De las líneas 15 a la 19 se declaran las variables globales que serán accesibles por todos los métodos estáticos de la clase como ser el puerto de escucha, el tamaño del buffer y el verdadero buffer donde utilizamos el tamaño definido para crearlo, recordemos que una vez definido el arreglo que actuará de buffer no podemos expandirlo o reducirlo, por ello el cálculo previo para estimar su máxima cantidad de datos. También se define el tiempo en milisegundos que esperaremos antes de mostrar en pantalla el siguiente teléfono.

A partir de la línea 21 comienza el método principal, el cual se ejecutará paso seguido de inicializar las variables globales. Revisemos la línea 23, la declaración de la variable ***skt***, no estamos inicializando el objeto de la biblioteca, sino que lo estamos apuntando a un objeto especial de tipo **nulo**, este objeto llamado ***null*** es un objeto vacío que no posee propiedades de ningún tipo y cada vez que queramos operar sobre el producirá un error de puntero nulo, entonces ¿porqué lo necesitamos?. El compilador no es lo suficientemente inteligente como para detectar que lo inicializaremos en un bloque de código futuro y necesitamos declararlo en este punto de nuestro método principal, porque todo lo que se declara en bloques especiales como try{}catch(){} es local y desaparece fuera de ellos.

En la línea 25 declaramos el objeto de la biblioteca ServerSocket que escuchará y devolverá un objeto del tipo Socket, usamos el mismo objeto **nulo** para inicializarlo.

Desde la línea 27 a la 32 intentamos crear el objeto ServerSocket y lo que es más importante, detectar si hubo o no un error por parte del sistema ya que no podremos continuar si se produce un error, como podemos ver en nuestro bloque de control de errores salimos si el sistema no nos permite abrir dicho puerto, en el caso que tengamos algún tipo de protección de red tendremos que desactivarla temporalmente para poder trabajar.

Entre las líneas 34 y 39 nos encargamos de esperar una conexión remota, nuevamente es necesario capturar las posibles excepciones de error que puedan producirse, es obligatorio por el compilador manejar las excepciones declaradas por ServerSocket, como podemos ver, la función accept(); de ServerSocket aceptará una conexión y esa conexión la almacenaremos en el apuntador skt, teniendo ahora correctamente inicializado dicho apuntador podemos usar las características de la conexión.

A partir de la línea 42 comenzamos la operación de lectura de la conexión de red y de la impresión en pantalla de los teléfonos.

Ya habíamos visto los objetos del tipo ***InputStream*** para la lectura de texto desde el teclado, en este caso lo usaremos para la lectura desde la conexión, solo que para practicar, utilizaremos un objeto de la misma familia pero del tipo ***ObjectInputStream*** que nos permitiría enviar objetos a través del canal o flujo de datos.

El objeto **skt** provee la función getInputStream(); que nos devolverá el Objeto del tipo InputStream que necesita el objeto de la biblioteca del tipo ***ObjectInputStream*** para ser construido.

En la línea 45 declaramos una variable de tipo entero largo (long) **timer** que utilizaremos para medir el tiempo y así crear nuestro temporizador, le damos un valor inicial.

En la línea 46 declaramos una variable booleana **timer_on** que nos servirá de bandera para saber el estado del temporizador, si es que está activo (true) o parado (false). Lo inicializamos en parado porque no tenemos teléfonos hasta que la secretaria los comience a escribir.

En la línea 47 se comienza un bloque del tipo "mientras", while(){} el cual deseamos que sea infinito porque leeremos datos hasta que la secretaria envíe la señal de fin, pero mientras tengamos teléfonos para mostrar en el buffer no podremos dejar de mostrarlos, así que para mejor comprensión del código, el bloque while(**true**){} se ejecutará por siempre a menos que el programa decida terminar.

Revisemos el código de la línea 48 a la 52, en este bloque comparamos dos condiciones

importantes para decidir si el programa debe salir o no, la primera es una condición doble, por eso está encerrada entre paréntesis, es decir *si la conexión está cerrada y la cantidad de elementos del buffer es inferior a 1* o sucede que *el elemento cero del buffer es el número -1 que indica la secretaria que terminó* entonces nos vamos porque es el último elemento y no es necesario mostrarlo.

El siguiente bloque desde la línea 54,55 almacenamos un número entero desde la conexión de red **datos**, sólo si es que hay datos disponibles, eso lo revisamos con la función available(); que no se quedará esperando, si existen o no datos disponibles de todas maneras el programa sigue su curso.

El timer, entre las líneas 58 a la 65, funciona de la siguiente manera, cuando queremos comenzar a medir el tiempo, almacenamos dentro de la variable timer la hora actual medida en milisegundos, ese dato lo obtenemos con la función del objeto system, ***currentTimeMillis()***, si al timer, le sumamos la cantidad de tiempo que queremos esperar, ese valor siempre será más grande que la hora actual, hasta que se pase el tiempo, eso es lo que comparamos en el bloque if(){}, caso contrario nos preparamos para ver si es necesario encender el timer y mostrar teléfonos, que sólo sucederá si hay más de 1 elemento en el buffer, recordemos que en buff_elem almacenamos la cantidad de elementos de nuestra memoria temporal.

En las líneas 67,68 incluimos otro bloque de control de errores obligatorio porque utilizaremos otra función de la biblioteca de Java que pertenece a la clase Thread (hilo) esa función detiene la ejecución del programa una cierta cantidad de milisegundos, el uso de este pequeño retardo de 100 milisegundos es para no sobrecargar el procesador, ya que estaríamos ejecutando nuestro bucle while(true){} infinidades de veces en 1 único segundo lo cual no es necesario ya que la secretaria no es tan veloz y dejamos libre el procesador para el sistema operativo.

En la línea 69 termina el bloque eterno while(true){} y el bloque de control de errores que lo contiene, dado el caso que se produzca un error dentro del bloque será necesario terminar el programa.

Los métodos auxiliares:

```
public static void put_tel(int tel){
```

Esta función agregará un teléfono en la memoria temporal, en nuestro buffer, haciendo una cierta detección de errores, en el caso que el buffer no sea lo suficientemente grande como para almacenar el próximo dato, en este caso el programa se sale con error.

Si esto no sucede, la línea más importante de esta función es:

```
buffer[buff_elem++]=tel;
```

Donde como vimos al principio de este libro el signo ++ a la derecha de buff_elem simboliza que primero se utilizará el valor y después se incrementará, entonces en una sola línea representamos las dos operaciones buffer[buff_elem]tel; buff_elem++;

```
public static int get_tel(){
```

Esta función extraerá un elemento del buffer, pero al mismo tiempo que lo extrae debe posicionar los demás elementos hacia el inicio del arreglo.

Tampoco hacemos la comprobación de que no haya elementos en el buffer porque la hacemos en el código cada vez que vemos si hay teléfonos para mostrar, aunque deberíamos por buena práctica porque si este es un prototipo y nuestro programa irá creciendo, entonces tal vez se nos escape revisar.

Almacenamos de manera temporal el teléfono próximo antes de recorrer los datos:

```
int tel=buffer[0];
```

Después de eso quitamos uno al apuntador de elementos, y lo quitamos antes porque para subir los datos sumaremos uno al índice que recorremos para acomodar los datos y tendríamos que estar

revisando que el índice no supere buff_elem-1, como esa condición se compararía cada vez que pasamos por un valor del índice estaríamos ejecutando innecesariamente una resta que podríamos hacer antes.

Una vez terminado devolvemos el teléfono que almacenamos antes que desapareciera.

Nota: Es incorrecto el manejo de error en la función put_tel, ya que en Java existe un modelo de excepciones que deben ser lanzadas y capturadas como en cualquier bloque de control de errores, si no corroboramos el índice Java producirá una IndexOutOfBoundException que de todas maneras terminará el programa.

Secretaria:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. import java.net.*;
4. /**
5.  * <p>Título: Aprendiendo Java</p>
6.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
7.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
8.  * <p>Empresa: COMPUNAUTA</p>
9.  * @author Gustavo Guillermo Pérez
10. * @version 2006.01.01
11. */
12.
13.public class Cap3_sock_sec {
14.//Declaramos unas variables globales a este tipo de datos
15.public static int PORT=4567;
16.public static String HOST="127.0.0.1";
17.
18.  public static void main(String[] args) {
19.    System.err.println("Intentando conectar con la telefonista");
20.    Socket skt=(Socket)null;
21.    try {skt = new Socket(HOST, PORT);}
22.    catch (Exception ex) {
23.      System.err.println("La telefonista no está en línea");
24.      System.exit(-1);
25.    }
26.
27.    int tel;
28.    BufferedReader teclado=new BufferedReader(new
   InputStreamReader(System.in));
29.    try {
30.      ObjectOutputStream datos = new
   ObjectOutputStream(skt.getOutputStream());
31.      System.err.println("Nos conectamos con la telefonista:"+HOST
   +":"+PORT);
32.      System.err.println("Ingrese números -1 termina");
33.      while (true){
34.      if((tel=leerInt(teclado))==-1){
35.        System.err.println("Programa terminado");
36.        datos.writeInt(-1);
37.        datos.flush();
38.        datos.close();
39.        skt.close();
40.        System.exit(0);
41.      }else{
42.        datos.writeInt(tel);
43.        datos.flush();
44.      }
45.    }//fin de la lectura eterna
46.    }catch (IOException ex1) {ex1.printStackTrace(System.err);}
```

```
47.
48.  }//fin del método principal
49.
50.//Funciones o métodos auxiliares
51.public static int leerInt(BufferedReader buff){
52.    int lee=0;
53.    boolean error;
54.    do {
55.      error=false;
56.      try {lee = Integer.parseInt(buff.readLine());}
57.      catch (NumberFormatException ex) {
58.  System.err.println("Entrada erronea, repetir:?");
59.        error=true;}
60.      catch (Exception ex){ex.printStackTrace(System.err);}
61.    }  while (error);
62.  return lee;
63.  }//final de la funcion leer
64.
65.}//final de la clase
```

Descripción del funcionamiento:

En las líneas 15 y 16 declaramos las variables globales que almacenarán el puerto y la dirección de red donde nos conectaremos, en este caso la dirección 127.0.0.1 es universal y permite conectarse a la misma máquina así que podremos ejecutar el programa de la secretaria y la telefonista en la misma PC en consolas de comandos diferentes, si estamos en el laboratorio con más de 1 PC en RED, entonces en ese número pondremos el que nos comente el administrador del sistema para conectarnos a la PC que correrá la telefonista.

En la línea 18 comienza nuestro método principal y de la misma manera que en el programa anterior definimos el Socket (skt) como un elemento vacío sin inicializar, ya que estamos obligados a revisar errores.

Entre las líneas 21 a 25 comprobamos que se pueda hacer una conexión al puerto y dirección de red indicados, puede suceder que el error no sea exactamente que la telefonista no está en línea y que el sistema operativo esté denegando el acceso a conectarse al exterior, por ello revisar el firewall del sistema o los permisos necesarios.

Si nos pudimos conectar el programa siguió adelante y procedemos a crear un objeto BufferedReader para manipular la entrada por teclado y declaramos una variable tel que almacenará un teléfono para enviar.

En la línea 29 comienza el bloque donde abriremos la conexión de datos remota por medio de un flujo de datos del tipo ***ObjetOutputStream***.

Como en el componente anterior utilizamos un bucle eterno para la ejecución del programa que saldrá por otros mecanismos. (línea 33).

En la línea 34, asignamos el valor que se lee por teclado a la variable tel, ese segmento está entre paréntesis porque porque ese mismo valor asignado lo compararemos antes de proseguir, podríamos haber hecho primero la asignación y después la comparación.

Si la secretaria escribió -1 enviamos el -1 a la telefonista para avisarle que terminamos, (línea 36), la orden ***datos.fluxh();*** vacía los datos de memoria RAM al flujo de datos inmediatamente y en las líneas 38 y 39 cerramos debidamente el flujo de datos y la conexión de red antes de salir sin producir error, salir con el código 0 es no producir error.

En la línea 41 comienza el bloque de datos que se ejecutará caso contrario a que la secretaria ingrese un -1. Donde podemos ver que escribimos en el flujo de datos el número entero correspondiente al teléfono y obligamos la escritura inmediata con la orden ***flush();***.

Termina el bucle etenro while(true){} termina el bloque de control de errores y el método principal, a continuación están la función auxiliar leerInt que no explicaremos porque es la misma que vimos en el capítulo anterior.

> ***Nota: Veremos con más detalles el uso de sockets durante el transcurso del libro, no ahondaremos en ello ahora. Queda como ejercicio para el lector producir el error de Buffer Overrun y revisar en la documentación electrónica los métodos y objetos utilizados de la biblioteca de Java.***

Implementación del buffer tipo LIFO (La pila, último en llegar es primero en salir)

Modifiquemos el ejemplo anterior para que ahora sean una bibliotecaria y su asistente, la bibliotecaria recibirá libros nuevos para catalogar y los enviará a apilar en el escritorio de la asistente, la asistente tendrá un minuto para leer el nombre de la etiqueta que envió la bibliotecaria a través de la red y ordenarlo, es visto que los libros ahora saldrán en forma de pila y no cola de espera, el límite será la altura de la asistente que hace las tarjetas sentada, pero no lo calcularemos :p.

Libros en total 24, se envía el contenido de la tarjeta en modo texto.

Asistente:

```
gus@gusgus ~$ java com.compunauta.aprendiendojava.Cap3_lifo_asis
Escuchando el puerto:4567
Esperando conexión...
Conectado... Esperando títulos
Libro:Introducción a la física
No hay más, nos vamos cuando terminemos...
Libro:Aprendiendo Java
Libro:Lectura I
Libro:Asambleas
Ya no es necesario esperar, terminado...
gus@gusgus ~$
```

Bibliotecaria:

```
gus@gusgus ~$ java com.compunauta.aprendiendojava.Cap3_lifo_bib
Intentando conectar con la asistente
Nos conectamos con la asistente:127.0.0.1:4567
Ingrese Títulos (línea vacía termina)
Introducción a la física
Asambleas
Lectura I
Aprendiendo Java
Programa terminado
gus@gusgus ~$
```

Asistente:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. import java.net.*;
4. /**
5.  * <p>Título: Aprendiendo Java</p>
6.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
7.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
8.  * <p>Empresa: COMPUNAUTA</p>
9.  * @author Gustavo Guillermo Pérez
10. * @version 2006.01.01
```

```
11. */
12.
13.public class Cap3_lifo_asis {
14.//Declaramos unas variables globales a este tipo de datos
15.public static int PORT=4567;
16.public static int BUFF_SIZE=24;
17.public static int TIMER_SLEEP=60*1000; //60sx1000ms
18.public static int buff_elem=0;
19.public static String[] buffer=new String[BUFF_SIZE];
20.
21.  public static void main(String[] args) {
22.    //Declaramos la variable socket (será un puntero a objeto)
23.    Socket skt=(Socket)null;
24.    //Declaramos vacío el servidor de sockets para inicializarlo
25.    ServerSocket Ss=(ServerSocket)null;
26.
27.    //Tratamos de escuchar el puerto definido por la variable PORT
28.    System.err.println("Escuchando el puerto:"+PORT);
29.    try {Ss = new ServerSocket(PORT);}
30.    catch (IOException ex) {
31.      System.out.println("El sistema no permite abrir el puerto");
32.      System.exit(-1);}
33.
34.    //Si no ocurrió error arriba entonces esperamos a la secretaria
35.    System.err.println("Esperando conexión...");
36.    try {skt = Ss.accept();}
37.    catch (IOException ex1) {
38.      ex1.printStackTrace(System.err);
39.      System.exit(-1);}
40.
41.    //Si no ocurrió error arriba la secretaria está lista para enviar
42.    System.err.println("Conectado... Esperando títulos");
43.    try {
44.      BufferedReader datos = new BufferedReader(new
   InputStreamReader((skt.getInputStream())));
45.      long timer=0;
46.      boolean timer_on=false;
47.      boolean ultimo=false;
48.      while (true){
49.        if(!ultimo && (skt.isClosed() || ((buff_elem>0) &&
   buffer[buff_elem-1]!=null && buffer[buff_elem-1].equals("fin")))){
50.        //Terminamos el programa si la bibliotecaria terminó
51.        System.err.println("No hay más, nos vamos cuando terminemos...");
52.        //el libro fin no se debe guardar es el aviso
53.        buff_elem--;
54.        ultimo=true;
55.        }
56.        if(ultimo && (buff_elem==0)){
57.        System.err.println("Ya no es necesario esperar, terminado...");
58.        System.exit(0);}
59.        //si hay títulos los guardamos
60.        if(!ultimo && datos.ready()){
61.        put_tit(datos.readLine());}
62.        if(timer_on){
63.          //si el timer funciona no hacer nada, si se pasó pararlo
64.          if ((timer+TIMER_SLEEP)<System.currentTimeMillis())
   {timer_on=false;}
65.        }else{
66.          //Si el timer está apagado, mostramos un tel si es que hay
67.          if (buff_elem>0){System.out.println("Libro:"+get_tit());
68.          //Encendemos el timer y guardamos la hora en que empezó
```

```
69.           timer_on=true;
70.           timer=System.currentTimeMillis();}
71.         }
72.         //Pausamos 100ms para no sobrecargar el procesador
73.         try {Thread.sleep(100);}
74.         catch (InterruptedException ex3) {}
75.       }//fin del bloque eterno
76.     }catch (Exception ex2) {
77.       ex2.printStackTrace(System.err);
78.       System.exit(-1);
79.     }
80.
81.   }//fin del método principal
82.
83.
84.//Funciones o métodos auxiliares
85.public static void put_tit(String tit){
86.//Si se supera el espacio producir un error
87.if (BUFF_SIZE<(buff_elem+1)){
88.System.err.println("Buffer overrun: El buffer se llenó demasiado rápido");
89.System.exit(-1);}
90.//guardamos el tel y aumentamos en uno el contador
91.buffer[buff_elem++]=tit;
92.}
93.
94.public static String get_tit(){
95.   //quitamos uno al contador de elementos
96.   //devolvemos el último libro
97.   return buffer[--buff_elem];
98.}
99.
100.}//final de la clase
```

Descripción del funcionamiento:

La declaración de variables globales a la clase o tipo de datos se realiza entre las líneas 15 y 19.

El método principal comienza en la línea 21, y declaramos los objetos del tipo Socket y ServerSocket como en el apartado anterior.

Desde la línea 28 a la 32 intentamos abrir el puerto para escuchar conexiones entrantes, si sucede un error nos salimos.

Si no hay errores, entre la línea 35 y 38 esperamos una conexión entrante con su respectivo bloque de control de errores.

Como la función ***accept();***detendrá el programa hasta que arribe una conexión, si estamos ejecutando el bloque principal entre las líneas 43 y siguientes, es porque se recibió una conexión. En el respectivo bloque de control de errores iniciamos el flujo de datos que usaremos para ir recibiendo renglones, en este caso usamos el mismo tipo de objeto que nos permite leer líneas desde el teclado, solo que ahora las leeremos desde la conexión de red.

Declaramos algunas cuantas variables para crear nuestro temporizador de 1 minuto igual que en el ejemplo anterior, y definimos una variable **ultimo** que nos avisará cuando los libros son todos.

En este caso la bandera que usamos para saber que hemos terminado por parte de la bibliotecaria es la palabra clave “fin”, y por supuesto revisamos que la conexión no esté cerrada, que haya elementos y todo eso antes de proceder a retirar un libro del buffer y enseñarlo en pantalla. La bandera binaria **ultimo** solo se activará en la condición anterior para asegurar que seguiremos mostrando títulos mientras, es una variación respecto del ejemplo anterior para ver otras maneras de resolver algo similar.

Si, el ultimo elemento ya llegó y no hay más en la memoria temporal, entonces nos salimos del programa. (líneas 56-58).

Si no es el ultimo elemento y hay datos disponibles en el flujo de datos de red, entonces leer una línea y ponerla en la memoria temporal.

Si el timer está encendido entonces procedemos igual que antes, el único cambio sustancial serán las funciones que guardan y extraen los datos del buffer.

La única función que cambia es la que quita elementos del buffer, que es mucho más simple que antes.

```
  return buffer[--buff_elem];
```

Donde estamos decrementando el contador de elementos antes (el -- está a la izquierda) y como sabemos que los arreglos se acceden desde el 0 a cantidad-1 entonces es correcto el resultado devuelto.

Bibliotecaria:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. import java.net.*;
4. /**
5.  * <p>Título: Aprendiendo Java</p>
6.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
7.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
8.  * <p>Empresa: COMPUNAUTA</p>
9.  * @author Gustavo Guillermo Pérez
10. * @version 2006.01.01
11. */
12.
13.public class Cap3_lifo_bib {
14.//Declaramos unas variables globales a este tipo de datos
15.public static int PORT=4567;
16.public static String HOST="127.0.0.1";
17.
18.  public static void main(String[] args) {
19.    System.err.println("Intentando conectar con la asistente");
20.    Socket skt=(Socket)null;
21.    try {skt = new Socket(HOST, PORT);}
22.    catch (Exception ex) {
23.      System.err.println("La asistente no está en línea");
24.      System.exit(-1);
25.    }
26.
27.    String titulo;
28.    BufferedReader teclado=new BufferedReader(new
   InputStreamReader(System.in));
29.    try {
30.      PrintWriter datos = new PrintWriter(skt.getOutputStream());
31.      System.err.println("Nos conectamos con la asistente:"+HOST+":"+PORT);
32.      System.err.println("Ingrese Títulos (línea vacía termina)");
33.      while (true){
34.      if((titulo=leerLinea(teclado)).length()==0){
35.        System.err.println("Programa terminado");
36.        datos.println("fin");
37.        datos.flush();
38.        datos.close();
39.        skt.close();
40.        System.exit(0);
41.      }else{
42.        datos.println(titulo);
```

```
43.         datos.flush();
44.       }
45.     }//fin de la lectura eterna
46.     }catch (IOException ex1) {ex1.printStackTrace(System.err);}
47.
48.  }//fin del método principal
49.
50.//Funciones o métodos auxiliares
51.public static String leerLinea(BufferedReader buff){
52.   try {return buff.readLine();}
53.   catch (Exception ex){ex.printStackTrace(System.err);}
54.   return "";
55.   }//final de la función leer
56.
57.}//final de la clase
```

Esta implementación es idéntica a la de la secretaria del ejemplo anterior, solo que para variar utilizamos otro tipo de Objeto para el flujo de datos.

Implementación de una Lista de datos.

En este caso la memoria temporal será una simple lista, se pretenderá que el usuario ingrese una lista de nombres y que en cualquier momento se pueda buscar, borrar, o agregar elementos.

Salida por pantalla:

```
gus@gusgus ~$ java com.compunauta.aprendiendojava.Cap3_lista
SELECCIONE UNA OPCIÓN:
1) Ingresar un elemento al listado
2) Listar los elementos de la lista
3) Borrar un elemento de la lista
0) Salir
opción?
1
Dato:?
Azul
opción?
1
Dato:?
Celeste
opción?
1
Dato:?
Caffe
opción?
1
Dato:?
Osos
opción?
2
Item[0]:[Azul]
Item[1]:[Celeste]
Item[2]:[Caffe]
Item[3]:[Osos]
SELECCIONE UNA OPCIÓN:
1) Ingresar un elemento al listado
2) Listar los elementos de la lista
3) Borrar un elemento de la lista
0) Salir
opción?
3
```

```
Item a borrar:?
2
SELECCIONE UNA OPCIÓN:
1) Ingresar un elemento al listado
2) Listar los elementos de la lista
3) Borrar un elemento de la lista
0) Salir
opción?
2
Item[0]:[Azul]
Item[1]:[Celeste]
Item[2]:[Osos]
gus@gusgus ~$
```

Ahora veamos el código:

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3.
4. /**
5.  * <p>Título: Aprendiendo Java</p>
6.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
7.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
8.  * <p>Empresa: COMPUNAUTA</p>
9.  * @author Gustavo Guillermo Pérez
10. * @version 2006.01.01
11. */
12.
13.public class Cap3_lista {
14. //Variables globales
15. public static int MAX=15;
16. public static String[] lista=new String[MAX];
17. public static int lista_elem=0;
18.
19.  public static void main(String[] args) {
20.  BufferedReader teclado=new BufferedReader(new
   InputStreamReader(System.in));
21.  int op=-1;
22.  while(true){
23.    switch (op){
24.      case 1:
25.        System.out.println("Dato:?");
26.        ingresa(leerLinea(teclado));
27.        break;
28.      case 2:
29.        listar();
30.        break;
31.      case 3:
32.        System.out.println("Item a borrar:?");
33.        borrar(opcion(teclado));
34.        break;
35.      case 0:
36.        System.out.println("Terminado..");
37.        System.exit(0);
38.        break;
39.    }
40.    if(op!=1)imprimir_menu();
41.    System.out.println("opción?");
42.    op = opcion(teclado);
43.  }//fin del bucle eterno
```

```
44.   }//fin del método principal
45.
46.   //Funciones auxiliares
47.   public static void ingresa(String dato){
48.     lista[lista_elem++]=dato;
49.   }
50.
51.   public static void listar(){
52.     for (int i=0;i<lista_elem;i++){
53.       System.out.println("Item["+i+"]:["+lista[i]+"]");
54.     }
55.   }
56.
57.   public static void borrar(int item){
58.     lista_elem--;
59.     for (int i=item;i<lista_elem;i++){
60.       lista[i]=lista[i+1];
61.     }
62.   }
63.
64.   public static void imprimir_menu(){
65.   System.out.println("SELECCIONE UNA OPCIÓN:");
66.   System.out.println("1) Ingresar un elemento al listado");
67.   System.out.println("2) Listar los elementos de la lista");
68.   System.out.println("3) Borrar un elemento de la lista");
69.   System.out.println("0) Salir");
70.   }
71.
72.   public static int opcion(BufferedReader buff){
73.   int lee=0;
74.   boolean error;
75.   do {
76.     error=false;
77.     try {return lee = Integer.parseInt(buff.readLine());}
78.     catch (NumberFormatException ex) {
79.System.err.println("Entrada erronea, repetir:?");
80.       error=true;}
81.     catch (Exception ex){ex.printStackTrace(System.err);}
82.   }  while (error);
83.return lee;
84.}//final de la funcion leer
85.
86.public static String leerLinea(BufferedReader buff){
87.   try {return buff.readLine();}
88.   catch (Exception ex){ex.printStackTrace(System.err);}
89.   return "";
90.   }//final de la funcion leer
91.
92.}//fin de la clase
```

En esta implementación agregamos un menú de usuario en pantalla que nos permitirá interoperar un poco mejor el programa para probar todas las opciones e incluso hacer experimentos y expandirlo con los ejemplos que seguirán a este.

Como antes entre las líneas 15 y 17 declaramos globalmente la lista, sus límites y el apuntador de elementos.

Nuestro método principal (línea 20) creará un objeto como antes hicimos para leer desde el teclado (o podría ser una fuente externa como la red en los ejemplos anteriores).

Definimos de manera local al método principal la variable **op** que almacenará la opción de menú que haya escogido el usuario (línea 21).

De la misma manera que antes ejecutamos indefinidamente el bloque while(true){} aunque podríamos igual que en todos los casos anteriores proponer una condición viable que también sea válida para terminar el programa como por ejemplo (op==0).

Cada cláusula ***case*** ejecutará la función correcta, para no producir una reimpresión excesiva molesta en la pantalla del menú de usuario sólo lo reimprimiremos si la opción de menú no fue la de agregar datos (línea 40).

En las líneas 41 y 42 imprimimos la leyenda para solicitar la opción. Podemos utilizar print en vez de println para que el ingreso de datos quede junto al signo de interrogación.

Función ingresa (línea 47), muy simple antes de aumentar en uno el contador de elementos se utiliza el valor del índice para almacenar el valor del dato que se desea ingresar a la lista.

Función listar (línea 51), listamos todos los elementos de la lista en pantalla indicando su índice en el arreglo de texto lista[].

Función borrar (línea 57), decrementamos en uno el contador de elementos y recorremos todos los elementos restantes a la posición que queremos eliminar.

Función imprimir_menu() (línea 64), sólo imprime en pantalla las opciones que compara el método principal para realizar acciones.

Función opción (línea 72), idéntica a leerInt que hemos estado viendo en todos estos ejemplos.

Función leerLinea (línea 86), idéntica a leerLinea de todos estos ejemplos anteriores.

Nota: a partir de aquí añadiremos funciones para experimentar con la lista, es ejercicio para el lector agregar a este ejemplo las entradas del menú y los correctos mecanismos para ejecutar esa funcionalidad en los bloques case del método principal.

Búsqueda de datos

Veremos tres maneras diferentes de encontrar un elemento en una lista de datos, el primero es el más lógico, el secuencial, el opuesto, aleatorio, y el binario cuando la lista lleva algún tipo de orden.

Búsqueda secuencial

La búsqueda secuencial es el equivalente a recorrer todos los elementos de la lista y compararlos del primero al último de manera que cuando encontramos el elemento terminamos la búsqueda.

```
public static String search(String patron){
  for (int i=0;i<lista_elem;i++){
    if(lista[i].indexOf(patron)!=-1) return lista[i];
  }
  return null;
}
```

Nota: Recordemos que lista_elem siempre almacenará la cantidad de elementos y al comprarar i<lista_elem estamos asegurándonos que jamás se llegará a este valor, ya que los arreglos se acceden desde 0 hasta lista_elem-1.

Búsqueda aleatoria, desordenar lista.

Este método es completamente probabilístico y es funcional cuando el método binario u otro método más eficaz no visto sea aplicable. Es factible si el acceso a los datos es extremadamente lento, es decir si la búsqueda secuencial podría tardar horas para encontrar el último entre unos cuantos y queremos probar suerte (ya que las probabilidades de la búsqueda secuencial son las mismas), también

sirve para desordenar una lista, por ejemplo para repartir “cartas mezcladas de un juego virtual”.

Ejemplo: En una situación donde la consulta en línea de archivos clasificados depende de personal humano para deducir la respuesta y la respuesta humana puede tardar minutos en encontrar por ejemplo un sello en una carpeta, la búsqueda se minimizaría de manera aleatoria si tenemos suerte, otro ejemplo sería buscar dentro de los archivos de una lista de archivos alojados en un servidor remoto completamente saturado y con escasas posibilidades de transferencia a alta velocidad, también minimizaríamos el tiempo de búsqueda de manera que en el peor de los casos nuestra respuesta estaría entre el 50% y el 100% de los últimos registros y en el mejor de los casos entre el 0% y el 50% de los primeros registros, de todas maneras es la misma probabilidad que la búsqueda secuencial así que solo el azar es el que influye.

Búsqueda desordenada de la lista:

```
1.   public static String buscar_desordenado(String patron){
2.     int[] indice=new int[lista_elem];
3.     int aleatorio;
4.     for (int i=0;i<lista_elem;i++) indice[i]=i;
5.     for (int i=lista_elem;i>0;i--){
6.       aleatorio=(int)(Math.random()*i);
7.       if(lista[indice[aleatorio]].indexOf(patron)!=-1) return
   lista[indice[aleatorio]]+":"+(lista_elem-i);
8.       for (int j=aleatorio;j<i-1;j++)indice[j]=indice[j+1];
9.     }
10.    return null;
11.  }
```

Fabricamos un índice para acceder a la lista, con la cantidad de elementos actual y no la máxima que es su capacidad, ya que será de manera temporal y local a la función, todos los objetos y variables declarados dentro de una función a menos que sean declarados estáticos se perderán y se liberará memoria del sistema.

Rellenamos el índice con los elementos desde el 0 al máximo menos uno. Extraemos al azar un elemento del índice y recorremos los demás hasta el que quitamos, así comparamos un elemento de la lista al azar y terminamos si lo encontramos.

Nota: El tipo de búsqueda no es exacto ya que usamos indexOf en vez de equals. Queda para el lector agregar esta función en el ejemplo anterior y probarla.

Desordenar la lista:

```
12.  public static void desordenar(){
13.    int[] indice=new int[lista_elem];
14.    String[] desordenado=new String[lista_elem];
15.    int des_elem=0;
16.    int aleatorio;
17.    for (int i=0;i<lista_elem;i++) indice[i]=i;
18.    for (int i=lista_elem;i>0;i--){
19.      aleatorio=(int)(Math.random()*i);
20.      desordenado[des_elem++]=lista[indice[aleatorio]];
21.      for (int j=aleatorio;j<i-1;j++)indice[j]=indice[j+1];
22.    }
23.    for (int i=0;i<lista_elem;i++) lista[i]=desordenado[i];
24.}
```

Para desordenar la lista estamos usando una lista temporal, con su respectivo contador e índice variable de elementos.

Nota: Queda como tarea para el lector optimizar esta función para que no sea necesaria la lista temporal que en caso de escasos recursos y enormes listas no sería algo

permitido, utilizando sólo 1 objeto temporal del tipo String para intercambiar valores y eliminar el uso del índice temporal.

Búsqueda Binaria (lista ordenada)

La búsqueda binaria es un método simple que se usa para encontrar datos en una lista ordenada, el mecanismo es el siguiente...

Si tenemos una lista de n datos, ordenados de mayor a menor y queremos encontrar uno en particular, haremos una búsqueda con dos índices, o sea almacenaremos el valor del elemento comparado más grande y más pequeño hasta arrinconar el resultado o llegar a un punto donde no existe tal elemento pero los dos mas cercanos son estos índices mayor y menor.

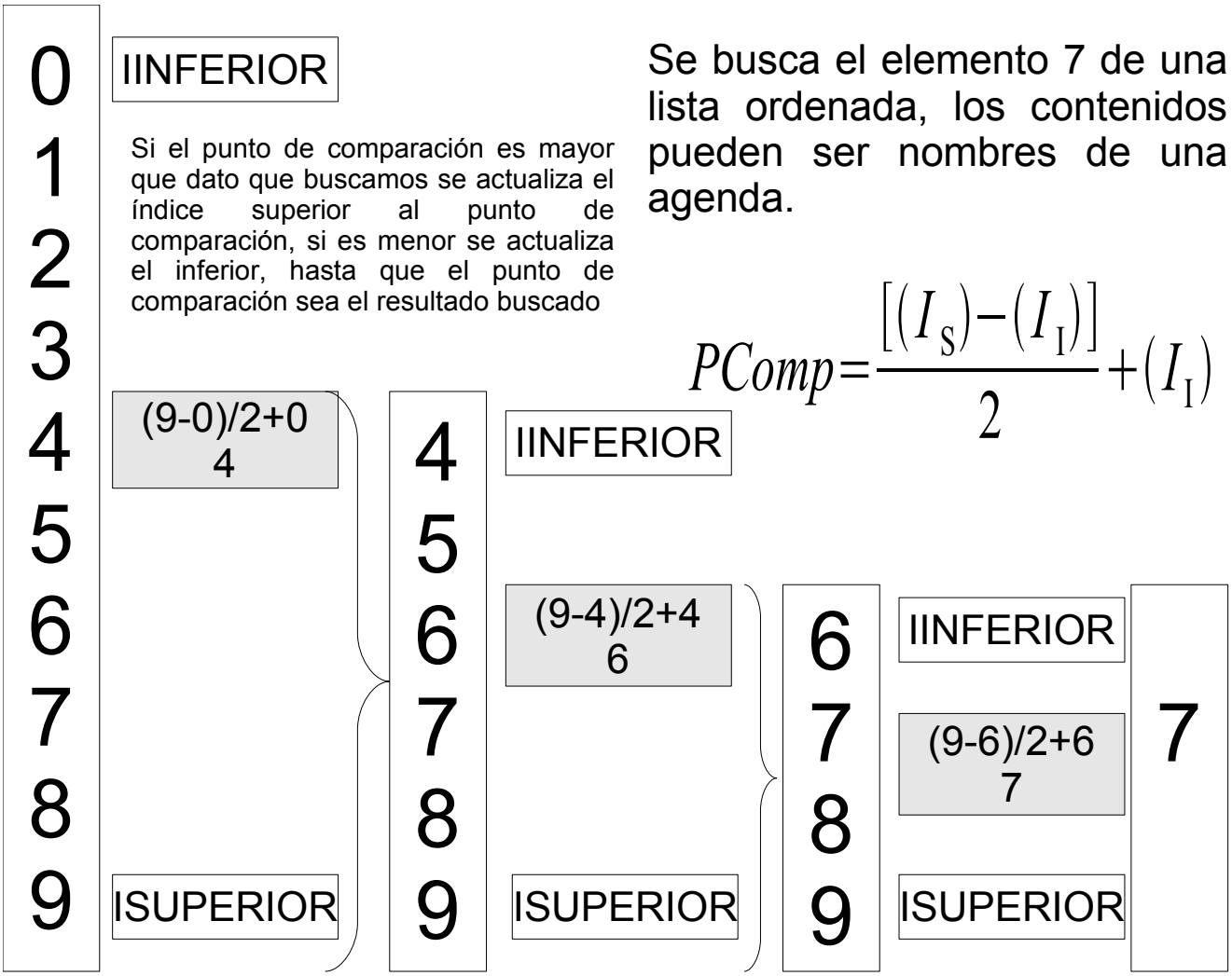

Ilustración 2: Búsqueda binaria en una lista ordenada

Veamos ahora un trozo de código para cuando la lista está ordenada de manera creciente y decreciente, estas dos funciones utilizan la fórmula que vimos en la ilustración anterior para calcular el índice superior e inferior, de esa manera y teniendo en cuenta que la operación de división será redondeada y sólo tendremos un número entero veamos como quedarían nuestras funciones.

Orden ascendente de Texto:

```
1. public static String buscar_ordenado_az(String inicio){
2.     int Is=lista_elem-1;
```

```
3.      int Ii=0;
4.      int cmp=0;
5.      int old_cmp=-1;
6.      int compare;
7.      while (cmp!=old_cmp){
8.        old_cmp=cmp;
9.        cmp=(Is-Ii)/2+Ii;
10.       compare=lista[cmp].compareTo(inicio);
11.       if(compare==0){return lista[cmp];}
12.       if(compare<0){Ii=cmp;}
13.       if(compare>0){Is=cmp;}
14.     }
15.     return lista[Is];
16.   }
```

Orden descendente de texto:

```
17.   public static String buscar_ordenado_za(String inicio){
18.     int Is=lista_elem-1;
19.     int Ii=0;
20.     int cmp=0;
21.     int old_cmp=-1;
22.     int compare;
23.     while (cmp!=old_cmp){
24.       old_cmp=cmp;
25.       cmp=(Is-Ii)/2+Ii;
26.       compare=lista[cmp].compareTo(inicio);
27.       if(compare==0){return lista[cmp];}
28.       if(compare>0){Ii=cmp;}
29.       if(compare<0){Is=cmp;}
30.     }
31.     return lista[Ii];
32.   }
```

En estas dos funciones iguales, sólo cambian las comparaciones, el problema en este ejemplo es la manera en la que la función de la biblioteca efectúa la comparación, podemos crear nuestras propias funciones de comparación y evitar usar las de la biblioteca, pero, la biblioteca suele tener funciones optimizadas al nivel del sistema operativo y no la máquina virtual de Java, por lo tanto las usaremos siempre que podamos.

> ***Nota: Queda para el lector agregar esta función al ejemplo de las listas y reconstruirla para el caso que los elementos sean números cualesquiera enteros en vez de objetos de texto.***

Métodos para ordenar listas

Veremos a continuación algunos pocos métodos para ordenar listas de datos, esto nos servirá para poder aplicar la búsqueda binaria sobre una lista de datos, aunque en la actualizad esto ya casi no se utiliza debido a que los motores de bases de datos ordenan los resultados (resultsets) de manera que ya no es necesario ordenar, se delega la tarea a programas optimizados para hacer eso de manera extraordinariamente rápida como MySQL compilado para el procesador de la PC.

Método de la burbuja o Método de Ordenación por Selección

Este método es antiguo y obsoleto, pero sirve para aprender, por su simpleza y porque para pocos elementos no consume muchos recursos, la idea es la siguiente: tenemos una lista de datos desordenada y comparamos los dos primeros y sólo esos dos los ordenamos de mayor a menor, el paso siguiente es comparar el que sigue con el anterior y así sucesivamente hasta llegar al final, si contamos las

comparaciones, fueron el total de los datos de la lista, menos una porque son de dos en dos. Con cada pasada por la lista un dato queda ordenado, y el nombre de burbuja es porque podríamos imaginarnos una burbuja que se lleva nuestro dato más grande hacia abajo o hacia arriba dependiendo como estemos recorriendo la lista y como estemos ordenando, si de mayor a menor o menor a mayor.

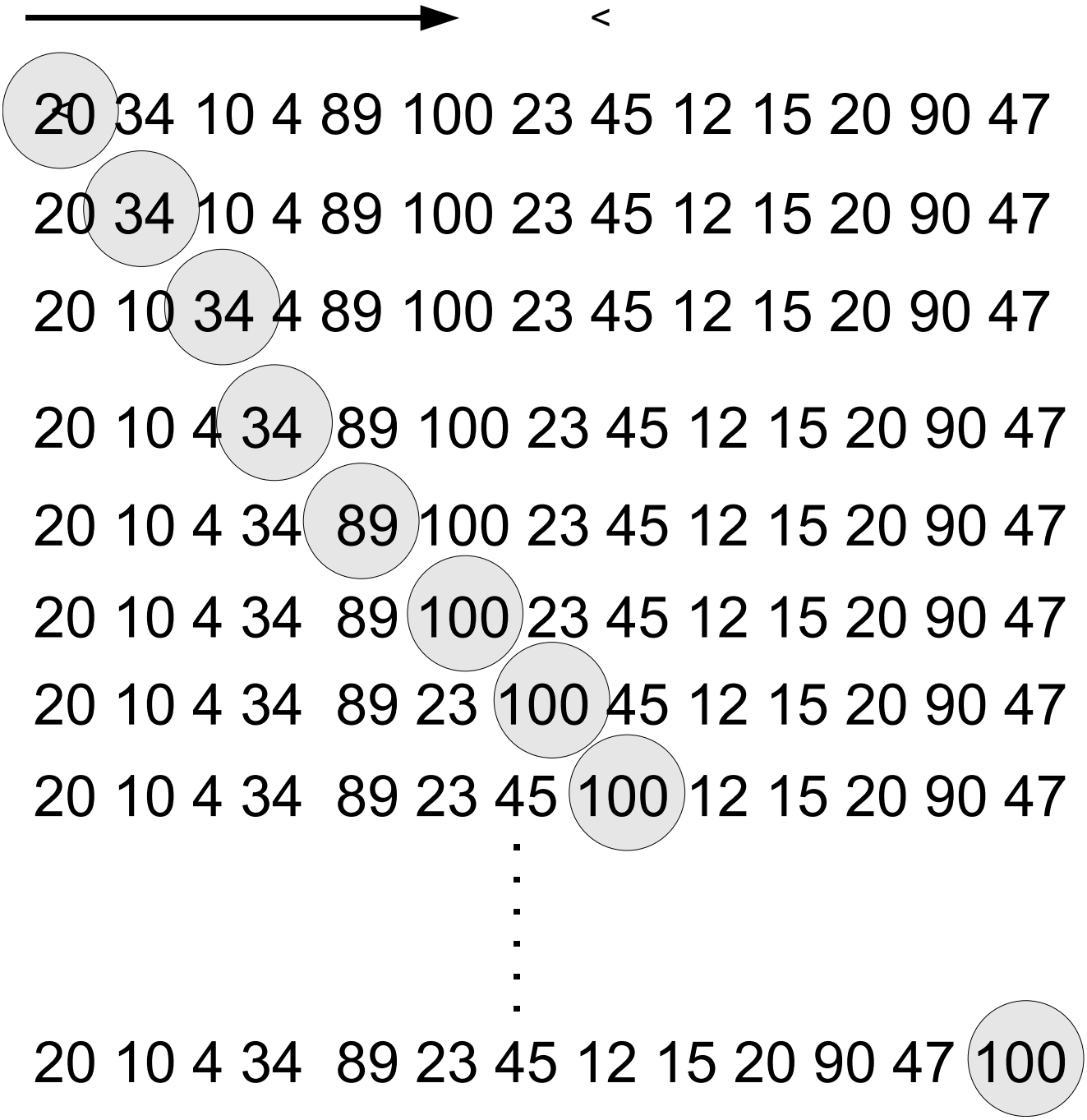

Ilustración 3: Método de la burbuja o Método de Ordenación por Selección, para ordenar datos

Como podemos ver, para que la lista quede ordenada en el ejemplo de la ilustración anterior, debemos repetir el procedimiento N-1 veces, pero también como podemos ver ya que el elemento más grande quedó ordenado, podemos reducir el espacio de la burbuja desde el primer elemento al N-1-i donde i es el número de veces que se ha recorrido la lista para ordenarla, de esa manera reducimos la cantidad de comparaciones, que hace la burbuja para quedarse con el elemento más grande.

Veamos un ejemplo:

```
1. public static void burbuja_mayormenor(int[] listado){
2.     int temporal;
3.     for (int j = 0; j < listado.length - 1; j++) {
4.       for (int i = 0; i < listado.length - 1; i++) {
5.         if (listado[i] < listado[i + 1]) {
6.           temporal = listado[i + 1];
7.           listado[i + 1] = listado[i];
8.           listado[i] = temporal;
9.         }
10.      }
11.    }
12.  } //final metodo ordenar burbuja mayor menor
```

la función descrita arriba ordena de mayor a menor los elementos de la lista, la lista es un arreglo de números enteros que se recorre de la misma manera que en la Ilustración 3 sólo que se compara de manera que el más pequeño sea el que se queda, en la linea 3 repetimos el proceso N cantidad de veces, ya que la burbuja ordena de 1 elemento por pasada y el proceso de recorrido empieza en la línea 4, en la línea 5 se hace la comparación y dado el caso se dan vuelta los elementos utilizando una variable temporal del mismo tipo que almacena el arreglo.

Ahora veamos como acotar los recorridos para reducir la cantidad de comparaciones:

```
13.public static void ordenar_burbuja_mayormenor(int[] listado){
14.    int temporal;
15.    for (int j = listado.length - 1; j > 0; j--) {
16.      for (int i = 0; i < j; i++) {
17.        if (listado[i] < listado[i + 1]) {
18.          temporal = listado[i + 1];
19.          listado[i + 1] = listado[i];
20.          listado[i] = temporal;
21.        }
22.      }
23.    }
24.  }//final metodo ordenar burbuja mayor menor
```

Este ejemplo es idéntico al anterior sólo que ahora la línea 3 no sólo se utiliza para contar la cantidad de veces que tenemos que recorrer el arreglo sino que va decreciendo para que el recorrido siempre sea un elemento menos, es decir el que quedó ordenado no se vuelve a comparar.

Nota: Queda como ejercicio para el lector construir las dos funciones inversas, las que ordenan de menor a mayor, y la que ordena de menor a mayor acotando las comparaciones.

Método QuickSort Recursivo

Este método para ordenar, es recursivo, si entendimos el ejemplo anterior, pudimos ver que la ejecución del algoritmo era secuencial, es decir 1 sola función hizo todo el trabajo en si misma.

Este algoritmo implementa la división en dos grupos del arreglo a ordenar y por lo tanto la misma operación se realiza con cada subgrupo que a su vez se vuelve a dividir, y como es la misma operación se llama a la misma función varias veces.

Esto tiene una desventaja, la plataforma que corre nuestra implementación debe producir una interrupción del proceso de ejecución para llamar a otra función, esto no tiene nada de malo excepto que cuando sean demasiados elementos la cantidad de veces que una función llamó a la misma función que llamó a la misma función puede ser tan larga que podría producir un agotamiento de recursos que ralentice o impida la ejecución de este tipo de algoritmos, en conclusión este método es muy difícil

implementarlo de manera secuencial pero la cantidad de comparaciones se reduce notablemente.{1}{2}

```
1.   public static void quicksort_menormayor(int[] listado,int i,int j){
2.     if (i>=j){return;}
3.     int p = dividiren_menormayor(listado,i,j);
4.     quicksort_menormayor(listado,i,p-1);
5.     quicksort_menormayor(listado,p+1,j);
6.   }//final del método principal de quicksort
7.
8.   public static int dividiren_menormayor(int[] listado,int i,int j){
9.     int p= i; i++;
10.    while (true){
11.      while (i < j && listado[i]<=listado[p]){i++;}
12.      while (i < j && listado[j]>=listado[p]){j--;}
13.      if (i == j) break;
14.      swap(listado,i,j);
15.    }
16.    if (listado[p]<listado[i]){i--;}
17.    swap(listado,p,i);
18.    return i;
19.  }//final del método que divide y ordena
```

En el código de ejemplo vemos que se declararon varias funciones, la función swap hace el intercambio con la variable temporal como en el método anterior, la función dividiren es la más importante de este procedimiento, es no solo la que divide sino la que intercambia los elementos que se van ordenando. Y por supuesto quicksort se llama así misma por cada subsegmento.

Cada segmento se recorre hacia arriba y hacia abajo saltándose los menores e iguales y mayores e iguales que el elemento inferior, cuando no hay nada que saltar, se intercambian los valores de manera que quedan por encima y debajo de *p* los mayores y los menores.

Ahora veamos la misma función pero ordenando de mayor a menor, como podemos ver sólo cambian los signos de comparación referentes a los contenidos del listado.

```
1. public static void quicksort_mayormenor(int[] listado,int i,int j){
2.     if (i>=j){return;}
3.     int p = dividiren_mayormenor(listado,i,j);
4.     quicksort_mayormenor(listado,i,p-1);
5.     quicksort_mayormenor(listado,p+1,j);
6.   }//final del método principal de quicksort
7.
8.   public static int dividiren_mayormenor(int[] listado,int i,int j){
9.     int p= i; i++;
10.    while (true){
11.      while (i < j && listado[i]>=listado[p]){i++;}
12.      while (i < j && listado[j]<=listado[p]){j--;}
13.      if (i == j) break;
14.      swap(listado,i,j);
15.    }
16.    if (listado[p]>listado[i]){i--;}
17.    swap(listado,p,i);
18.    return i;
19.  }//final del método que divide y ordena
```

Otros mecanismos para ordenar.

Existen otros mecanismos que no podemos tratar en este momento porque es necesario implementar objetos, así que ese será el próximo tema, algunos son, por inserción, por fusión, de shell y con un árbol binario.

Ejercicios

Los ejercicios del capítulo III son los últimos antes de proceder con objetos, es recomendable que todo aquello que se ha visto hasta el momento quede perfectamente asimilado ya que en el próximo tema los objetos comenzarán a ser la herramienta principal.

Ejercicio 3.1

Realizar un programa cuyo método principal pida 10 valores numéricos y los asigne a un arreglo, posteriormente imprimir en pantalla el resultado ordenado creciente, y decreciente utilizando el método de la burbuja, computar la cantidad de comparaciones realizadas entre elementos del arreglo y mostrarlas al final como estadística.

```
1. package com.compunauta.aprendiendojava;
2. import java.io.*;
3. /**
4.  * <p>Título: Aprendiendo Java</p>
5.  * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6.  * <p>Copyright: Copyright (c) 2006 www.compunauta.com</p>
7.  * <p>Empresa: COMPUNAUTA</p>
8.  * @author Gustavo Guillermo Pérez
9.  * @version 2007.10.01
10. */
11.
12.public class Cap3_ej1 {
13.  public static int comparaciones=0;
14.  public static void main(String[] args) {
15.    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
16.    int[] listado=new int[10];
17.    System.out.println("Ingrese 10 números, de a uno presionando ENTER");
18.    for(int i=0;i<10;i++){
19.      listado[i]=leerInt(br);
20.    }
21.
22.    burbuja_mayormenor(listado);
23.    for(int i=0;i<10;i++){
24.      System.out.println("NRO >:["+i+"] "+listado[i]);
25.    }
26.    System.out.println("Comparaciones:"+comparaciones);
27.    burbuja_menormayor(listado);
28.    for(int i=0;i<10;i++){
29.      System.out.println("NRO <:["+i+"] "+listado[i]);
30.    }
31.
32.  }
33.  public static int leerInt(BufferedReader buff){
34.    int lee=0;
35.    boolean error;
36.    do {
37.      error=false;
38.      try {lee = Integer.parseInt(buff.readLine());}
39.      catch (NumberFormatException ex) {
40.  System.out.println("Entrada erronea, repetir:?");
41.        error=true;}
42.      catch (Exception ex){ex.printStackTrace(System.err);}
43.    }  while (error);
44.  return lee;
45.  }//final de la función leer
46.
47.  public static void ordenar_burbuja_mayormenor(int[] listado){
```

```
48.     int temporal;
49.     for (int j = listado.length - 1; j > 0; j--) {
50.       for (int i = 0; i < j; i++) {
51.         comparaciones++;
52.         if (listado[i] < listado[i + 1]) {
53.           temporal = listado[i + 1];
54.           listado[i + 1] = listado[i];
55.           listado[i] = temporal;
56.         }
57.       }
58.     }
59.   }//final método ordenar burbuja mayor menor
60.
61.   public static void burbuja_mayormenor(int[] listado){
62.     int temporal;
63.     for (int j = 0; j < listado.length - 1; j++) {
64.       for (int i = 0; i < listado.length - 1; i++) {
65.         comparaciones++;
66.         if (listado[i] < listado[i + 1]) {
67.           temporal = listado[i + 1];
68.           listado[i + 1] = listado[i];
69.           listado[i] = temporal;
70.         }
71.       }
72.     }
73.   } //final método ordenar burbuja mayor menor
74.
75.   public static void ordenar_burbuja_menormayor(int[] listado){
76.     int temporal;
77.     for (int j = 0; j < listado.length - 1; j++) {
78.       for (int i = listado.length - 1; i > j; i--) {
79.         comparaciones++;
80.         if (listado[i] < listado[i - 1]) {
81.           temporal = listado[i - 1];
82.           listado[i - 1] = listado[i];
83.           listado[i] = temporal;
84.         }
85.       }
86.     }
87.   }//final método ordenar burbuja menor mayor
88.
89.   public static void burbuja_menormayor(int[] listado){
90.     int temporal;
91.     for (int j = 0; j < listado.length - 1; j++) {
92.       for (int i = listado.length - 1; i > 0; i--) {
93.         comparaciones++;
94.         if (listado[i] < listado[i - 1]) {
95.           temporal = listado[i - 1];
96.           listado[i - 1] = listado[i];
97.           listado[i] = temporal;
98.         }
99.       }
100.    }
101.  } //final método ordenar burbuja menor mayor
102.}
```

Línea 1, declaración del paquete, línea 2 importamos java.io para usar los métodos de lectura del teclado, Línea 12 declaración de la clase, línea 13 declaramos la variable estática a la clase comparaciones que es la que usaremos para computar la cantidad de comparaciones para ordenar el arreglo.

El programa en sí comienza en el método principal línea 14, línea 15 se construye un objeto *BufferedReader* para lectura por teclado, línea 16 se inicializa un arreglo de 10 elementos que será el tamaño del arreglo a trabajar.

Líneas 17 a 20 es la entrada por teclado de los números del arreglo utilizando la función leerInt(*BufferedReader*) utilizada en ejercicios anteriores.

Línea 22 se invoca a la función que ordena sobre el arreglo *listado* de números enteros, y en las líneas 23 a 26 se imprime en pantalla la información correspondiente y la estadística de las comparaciones. Línea 33 a 45 Función leer.

Línea 47 – 59 la función ordenar de mayor a menor con la optimización de acotación de índices, y las demás variantes con y sin optimización.

Ejercicio 3.2

Realizar un programa cuyo fin sea el del ejercicio 3.1 pero utilizando el procedimiento quicksort.

Ejercicio 3.3

Integrar con un pequeño menú en pantalla la selección del método que se utilizará para ordenar el arreglo reutilizando el código de los ejercicios anteriores 3.1 y 3.2, pero añadir al cómputo las comparaciones en total incluyendo las de los índices en bucles.

IV- Primeros Objetos como mecanismo de programación.

En este capítulo del libro comenzaremos a utilizar objetos para resolver problemas, la construcción eficiente del objeto también será parte del problema, recordemos que un objeto puede tener, métodos estáticos y dinámicos y propiedades estáticas y dinámicas, la selección de cada una dependerá del problema y del criterio utilizado.

Lista con punteros para ordenar datos.

Éste es otro método un poco más complejo para ordenar datos mientras se insertan, lo propondremos como una variante a los métodos vistos anteriormente pero construiremos un objeto que almacenará el dato, proveerá métodos para compararlo con otros objetos iguales y tendrá propiedades que nos permitirá conectarlo con otros objetos del mismo tipo. El objeto y el procedimiento irá haciéndose más complejo conforme avancemos en el tema.

Imaginemos una persona, tiene dos manos y cada mano puede unirse a otra persona, cada persona simboliza un objeto que almacena un dato o un conocimiento, entonces el objeto de nuestra lista puede vincularse por la izquierda y derecha o puede estar vinculado por solo uno de los dos lados, o ninguno en el estado inicial.

Cuando queremos insertar un elemento en nuestra lista, compararemos el nuevo de manera binaria recorriendo la lista como hemos visto anteriormente, y desconectaremos el lugar donde le corresponde ir uniendo su lado derecho e izquierdo con los elementos de la ruptura.

Este tipo de lista se maneja por apuntadores, vínculos entre objetos que no es lo mismo que tener un arreglo de datos que se puede acceder en cualquier momento a cualquier elemento por medio de su índice. En este caso es necesario ir recorriendo cada elemento para saber a quien está conectado y seguir la cadena de datos de derecha a izquierda y viceversa.{3}

Nuestro primer objeto

Para poder realizar la lista ordenada que mencionamos utilizaremos un objeto que representará el nodo que puede conectarse por izquierda y por derecha y que almacena el dato, conforme podamos introduciremos funciones extra para realizar comparaciones.

Propiedades

Necesitamos la propiedad derecha e izquierda que serán punteros o variables que almacenarán objetos del mismo tipo "nodo" que serán como las manos que conectan los objetos adyacentes.

Necesitamos la propiedad valor que almacenará una cantidad entera dentro del objeto, y esta cantidad se establecerá al construir el objeto.

```
1.package com.compunauta.aprendiendojava;
2.public class Nodo {
3.  public Nodo izquierda=null;
4.  public Nodo derecha=null;
5.  public int Dato;
6.  public Nodo(int Dato) {
7.    this.Dato=Dato;
8.  }
9.}
```

Ahora para hacer pruebas construiremos un método principal y como esta vez la información la almacenaremos en un arreglo sería interesante que tengamos un par de apuntadores principales que nos permitan acceder al primer y último objeto de manera inmediata, así que incluiremos esos apuntadores para trabajar. Utilizaremos también las funciones que hemos visto anteriormente para leer enteros del

teclado y recorreremos la lista del elemento inicial (el más pequeño) al elemento final (el más grande), después incluiremos algunas modificaciones para mejorar la capacidad de inserción acortando las comparaciones y más adelante rediseñaremos el método para tener mayor control sobre los elementos que no pertenecen a un arreglo en concreto.

```
1.package com.compunauta.aprendiendojava;
2.import java.io.*;
3./**
4. * <p>Título: Aprendiendo Java</p>
5. * <p>Descripción: Ejemplos del Libro Aprendiendo Java de Compunauta</p>
6. * <p>Copyright: Copyright (c) 2007 www.compunauta.com</p>
7. * <p>Empresa: COMPUNAUTA</p>
8. * @author Gustavo Guillermo Pérez
9. * @version 2007.10.02
10. */
11.
12.public class Nodo {
13.  public Nodo izquierda=null;
14.  public Nodo derecha=null;
15.  public int Dato;
16.  public Nodo(int Dato) {
17.    this.Dato=Dato;
18.  }
19.  public static Nodo ini=null; //inicio
20.  public static Nodo fin=null; //final
21.  public static int total=0;
22.  public static void main(String[] args) {
23.    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
24.    System.out.println("Ingrese 10 números, de a uno presionando ENTER");
25.    for (int i = 0; i < 10; i++) {
26.      addObject(leerInt(br));
27.    }
28.    printList();
29.  }
30.  public static void addObject(int Dato){
31.    Nodo dato=new Nodo(Dato);
32.    Nodo apuntador;
33.    total++;
34.    if (total==1){
35.      ini=dato;
36.      fin=ini;
37.    }else{
38.      if (fin.Dato <= Dato){
39.        fin.derecha=dato;
40.        dato.izquierda=fin;
41.        fin=dato;
42.        return;
43.      }
44.      if (ini.Dato >= Dato){
45.        ini.izquierda=dato;
46.        dato.derecha=ini;
47.        ini=dato;
48.        return;
49.      }
50.      apuntador=ini;
51.      while (apuntador.Dato < Dato){
52.        apuntador=apuntador.derecha;
53.      }
54.      dato.izquierda=apuntador.izquierda;
55.      apuntador.izquierda.derecha=dato;
```

```
56.       dato.derecha=apuntador;
57.       apuntador.izquierda=dato;
58.       return;
59.     }
60.   }
61.
62.   public static void printList(){
63.     Nodo apuntador=ini;
64.     int elementos=0;
65.     while (apuntador!=null){
66.       System.out.println("Elemento ["+(elementos++)+"]:"+apuntador.Dato);
67.       apuntador=apuntador.derecha;
68.     }
69.   }
70.   public static int leerInt(BufferedReader buff){
71.     int lee=0;
72.     boolean error;
73.     do {
74.       error=false;
75.       try {lee = Integer.parseInt(buff.readLine());}
76.       catch (NumberFormatException ex) {
77.   System.out.println("Entrada erronea, repetir:?");
78.         error=true;}
79.       catch (Exception ex){ex.printStackTrace(System.err);}
80.     }  while (error);
81.   return lee;
82.   }//final de la funcion leer
83.
84.}
```

Como podemos ver la función printList() recorre la lista para impresión en pantalla y la función leer renglones, es como la que hemos visto anteriormente, la que realiza la operación de inserción en la lista es addObject(int Dato) y tenemos en cuenta 4 casos.

1. El primer objeto es ini y fin y tiene nulas derecha e izquierda
2. Si no es el primer objeto y es más grande que todos unir a la derecha
3. Si no es el primer objeto y es más pequeño que todos unir a la izquierda
4. Si se encuentra en medio, recorremos desde el inicio hasta encontrar uno más grande.

Nota: Queda como ejercicio para el lector modificar el programa para recorrer la lista de menor a mayor o mayor a menor dependiendo del valor del dato a insertar.

Índice de tablas

Índice de ilustraciones

Referencias Recomendadas

1: , Ordenamiento de Arreglos, http://www.dcc.uchile.cl/~lmateu/Java/Transparencias/Compl/sort.htm
2: , Ordenación, http://www.dcc.uchile.cl/~cc30a/apuntes/Ordenacion/
3: Prof. Juan Andrés Colmenares, M.Sc. , Métodos de Ordenación Notas de Clase , http://www.ica.luz.ve/juancol/eda/ordenacion/index.html