Bases para una codificación óptima en Java

Objetivo:

Presentar los distintos recursos disponibles para mejorar la codificación en Java, sin profundizar en los mismos.

Temario:

- **Cómo hacer la prueba de unidad**
- **Uso de información de rastreo**
- **Técnicas propias de java para debug**
- **Cómo analizar el uso de recursos implícito en un Programa**
- **Metodología y optimización**

Prueba de Caja Blanca + Prueba de Caja Negra

IEEE ARGENTINA

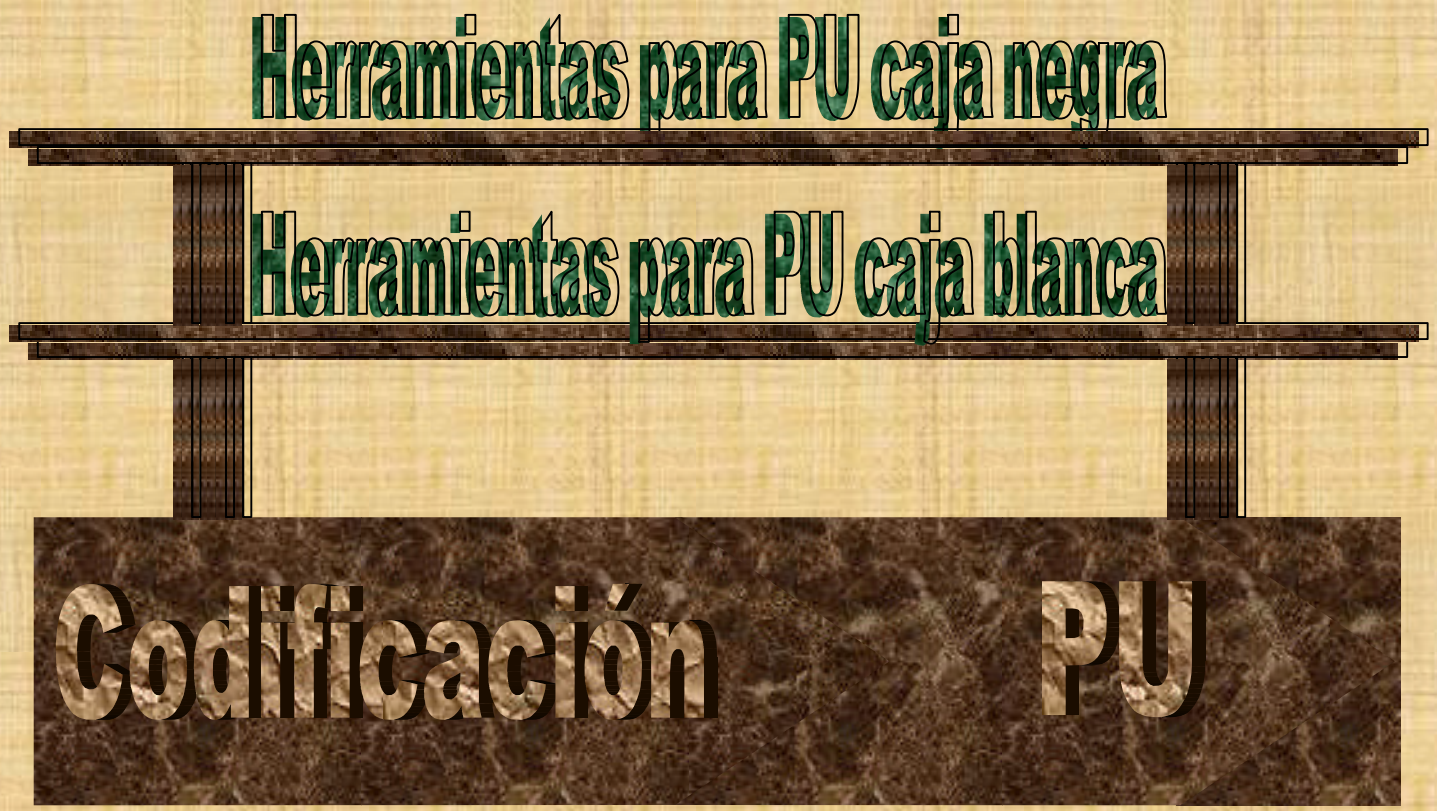
UP | Universidad de Palermo
Facultad de Ingeniería

Próximas charlas

SIP	(A. Popovski)	01/08/05 (19:00hs)
Webmining con Java	(D. López De Luise)	02/08/05 (19:00hs)
Análisis y auditoría de performance en internet	(A. Popovski)	22/09/05 (19:00hs)
Filtro semántico e IR	(N. Di Tada)	06/10/05 (19:00hs)
NN con Java	(D. López De Luise)	20/10/05 (19:00hs)
Java y el XML	(D. López De Luise)	27/10/05 (19:00hs)
La IA en Java	(D. López De Luise)	17/11/05 (19:00hs)
Cognitive Memory	(S. Piedrahita)	21/11/05 (19:00hs)
Webbrowsing con Java	(D. López De Luise)	24/11/05 (19:00hs)

upgrade@palermo.edu

sec.argentina@ieee.org


Herramientas para PU caja negra
Herramientas para PU caja blanca
Codificación PU

1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema open source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar p/mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

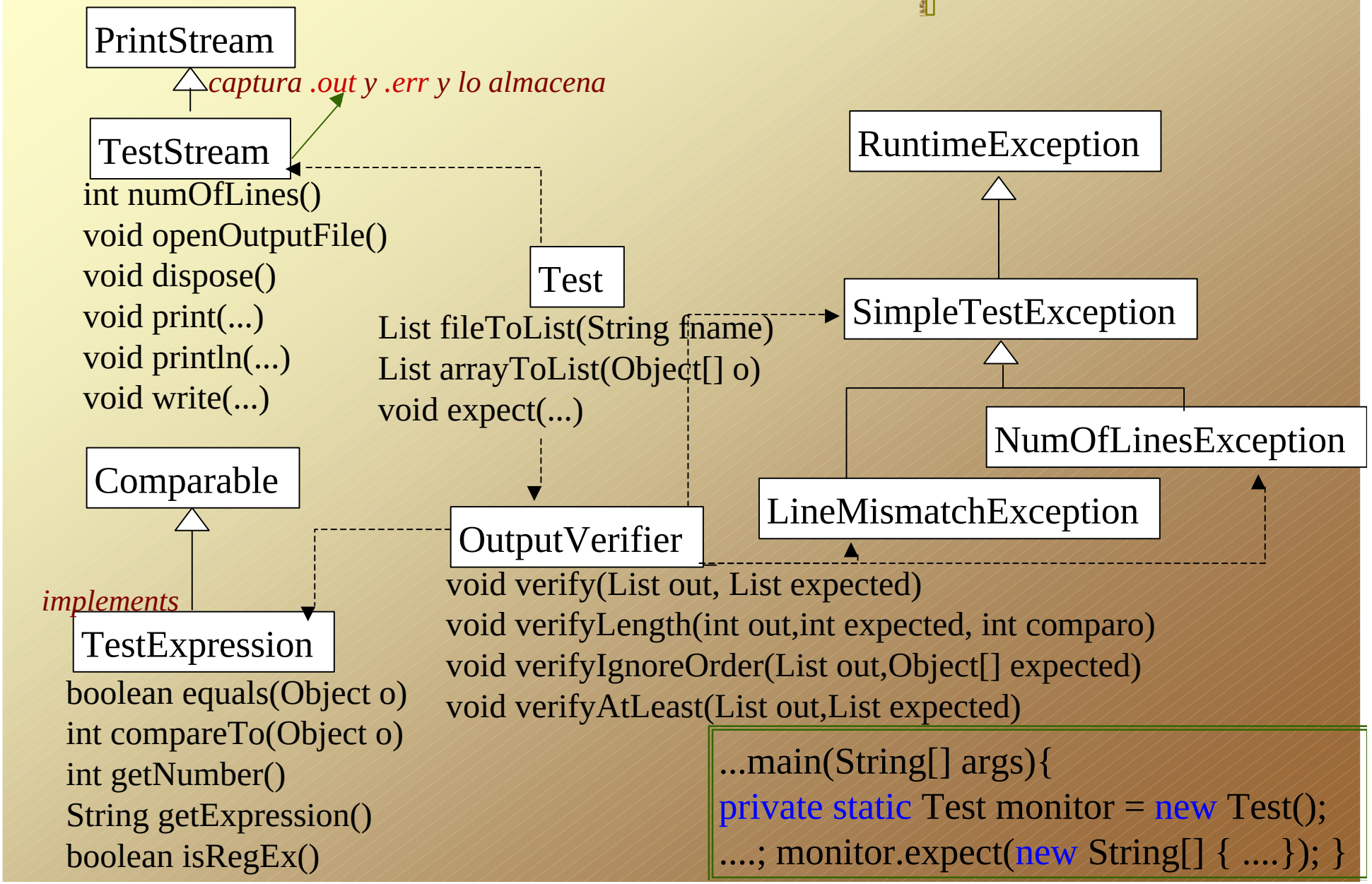
1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la corrección de una clase

1.a.PU con una herramienta hecha por nosotros



1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema open source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar para mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la corrección de una clase

1.b.II Provistas para Java: Junit

www.junit.org

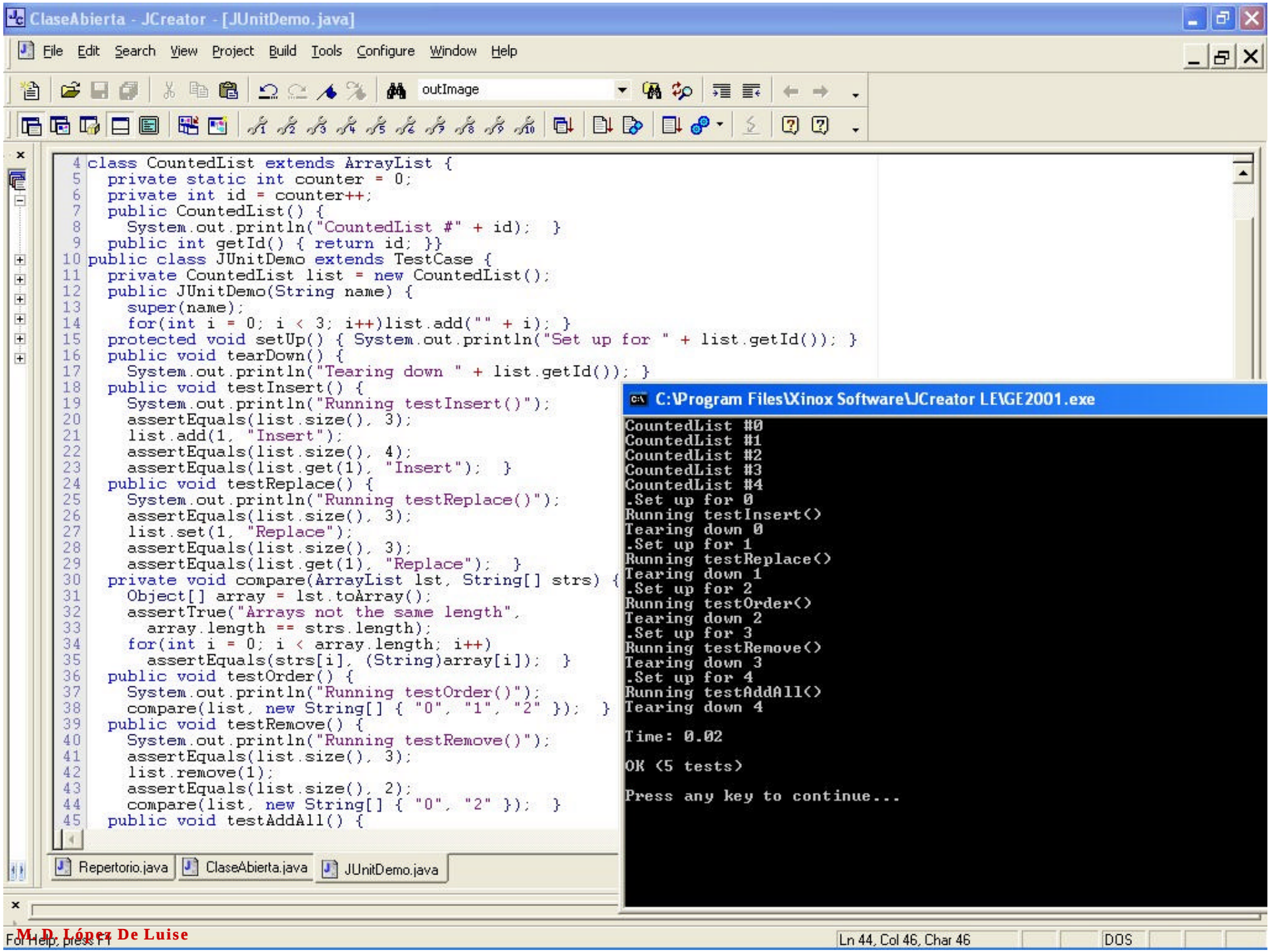
1.b.1 Provistas para Java: Junit
Erich Gamma & Kent Beck
www.junit.org

definición:

estándar para escribir tests de programas Java

características:

1. se escriben los test como subclases de TestCase
2. se escriben como métodos con signature:
   public void testXXXXXX ()
3. JUnit detecta y corre cada test inidivualmente eliminando efectos de arrastre entre tests consecutivos
4. setup()* inicializa el entorno para todos los test
5. tearDown() * cleanup necesario al terminar un test
6. el constructor debe pasar un String a la clase padre


ClaseAbierta - JCreator - [JUnitDemo.java]
File Edit Search View Project Build Tools Configure Window Help
outImage
4 class CountedList extends ArrayList {
5 private static int counter = 0;
6 private int id = counter++;
7 public CountedList() {
8 System.out.println("CountedList #" + id); }
9 public int getId() { return id; }}
10 public class JUnitDemo extends TestCase {
11 private CountedList list = new CountedList();
12 public JUnitDemo(String name) {
13 super(name);
14 for(int i = 0; i < 3; i++)list.add("" + i); }
15 protected void setUp() { System.out.println("Set up for " + list.getId()); }
16 public void tearDown() {
17 System.out.println("Tearing down " + list.getId()); }
18 public void testInsert() {
19 System.out.println("Running testInsert()");
20 assertEquals(list.size(), 3);
21 list.add(1, "Insert");
22 assertEquals(list.size(), 4);
23 assertEquals(list.get(1), "Insert"); }
24 public void testReplace() {
25 System.out.println("Running testReplace()");
26 assertEquals(list.size(), 3);
27 list.set(1, "Replace");
28 assertEquals(list.size(), 3);
29 assertEquals(list.get(1), "Replace"); }
30 private void compare(ArrayList lst, String[] strs) {
31 Object[] array = lst.toArray();
32 assertTrue("Arrays not the same length",
33 array.length == strs.length);
34 for(int i = 0; i < array.length; i++)
35 assertEquals(strs[i], (String)array[i]); }
36 public void testOrder() {
37 System.out.println("Running testOrder()");
38 compare(list, new String[] { "0", "1", "2" }); }
39 public void testRemove() {
40 System.out.println("Running testRemove()");
41 assertEquals(list.size(), 3);
42 list.remove(1);
43 assertEquals(list.size(), 2);
44 compare(list, new String[] { "0", "2" }); }
45 public void testAddAll() {
Repertorio.java
ClaseAbierta.java
JUnitDemo.java
C:\Program Files\Xinox Software\JCreator LE\GE2001.exe
CountedList #0
CountedList #1
CountedList #2
CountedList #3
CountedList #4
.Set up for 0
Running testInsert()
Tearing down 0
.Set up for 1
Running testReplace()
Tearing down 1
.Set up for 2
Running testOrder()
Tearing down 2
.Set up for 3
Running testRemove()
Tearing down 3
.Set up for 4
Running testAddAll()
Tearing down 4
Time: 0.02
OK (5 tests)
Press any key to continue...
For Help, press F1
Ln 44, Col 46, Char 46
DOS

1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema open source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar para mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la corrección de una clase

1.b.II. Provistas para Java: Ant

James Duncan Davidson

http://jakarta.apache.org/ant

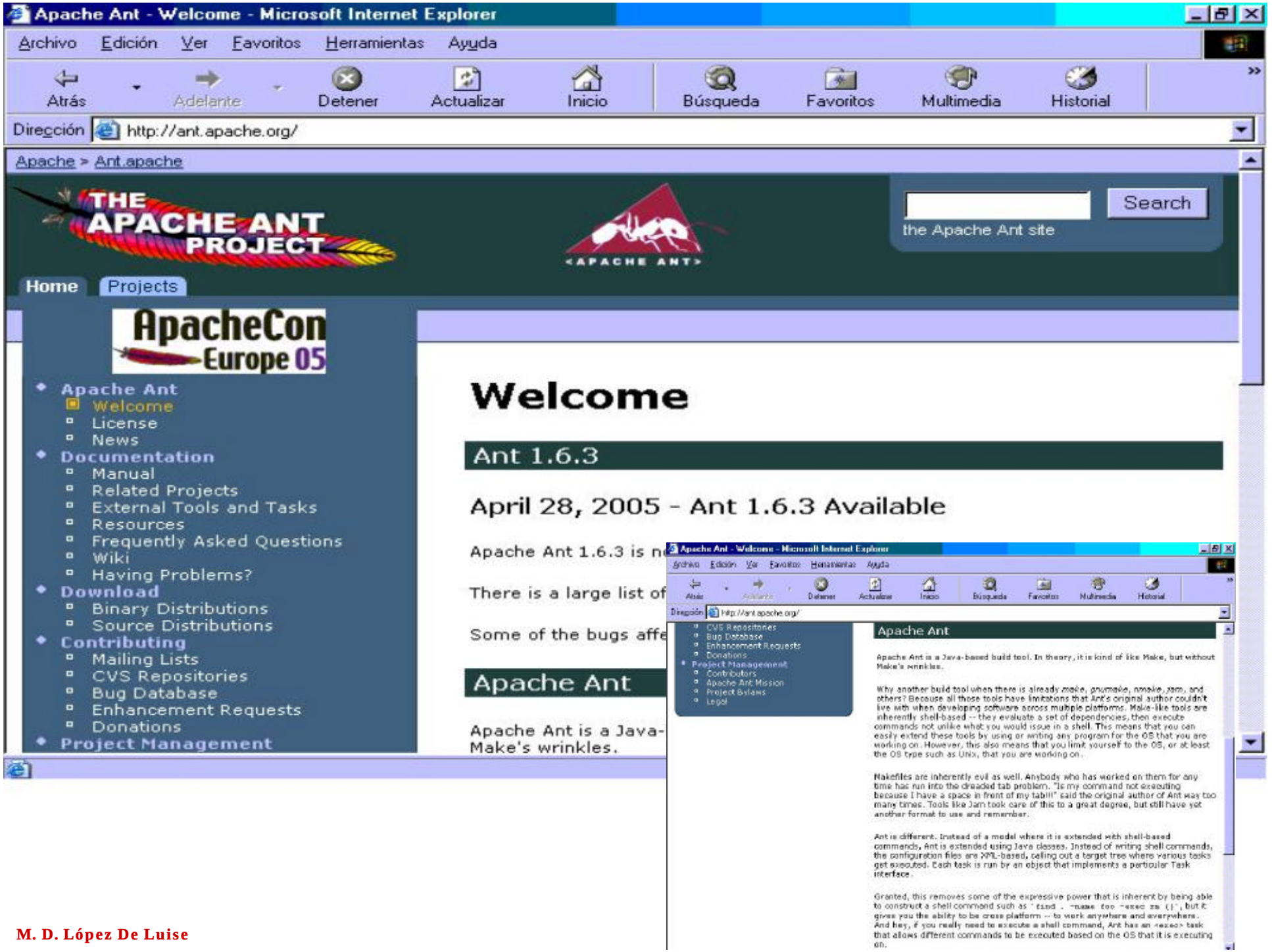
http://ant.apache.org

definición:

herramienta estándar para elaborar proyectos Java

características:

1. versión extendida, simplificada y cross-platform de make
2. basada en XML
4. resuelve los problemas específicos de la plataforma (EOL, separadores de directorios “/” Vs “\”, etc.)
5. extensible por el usuario
6. curva de aprendizaje aceptable


Apache Ant - Welcome - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Multimedia Historial
Dirección http://ant.apache.org/
Apache > Ant.apache
THE APACHE ANT PROJECT
<APACHE ANT>
Search
the Apache Ant site
Home Projects
ApacheCon Europe 05
Apache Ant
Welcome
License
News
Documentation
Manual
Related Projects
External Tools and Tasks
Resources
Frequently Asked Questions
Wiki
Having Problems?
Download
Binary Distributions
Source Distributions
Contributing
Mailing Lists
CVS Repositories
Bug Database
Enhancement Requests
Donations
Project Management
Welcome
Ant 1.6.3
April 28, 2005 - Ant 1.6.3 Available
Apache Ant 1.6.3 is n
There is a large list of
Some of the bugs affe
Apache Ant
Apache Ant is a Java-
Make's wrinkles.
Apache Ant
Apache Ant is a Java-based build tool. In theory, it is kind of like Make, but without Make's wrinkles.
Why another build tool when there is already make, gnumake, nmake, jam, and others? Because all those tools have limitations that Ant's original author couldn't live with when developing software across multiple platforms. Make-like tools are inherently shell-based -- they evaluate a set of dependencies, then execute commands not unlike what you would issue in a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on. However, this also means that you limit yourself to the OS, or at least the OS type such as Unix, that you are working on.
Makefiles are inherently evil as well. Anybody who has worked on them for any time has run into the dreaded tab problem. "Is my command not executing because I have a space in front of my tab!!!" said the original author of Ant way too many times. Tools like Jam took care of this to a great degree, but still have yet another format to use and remember.
Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface.
Granted, this removes some of the expressive power that is inherent by being able to construct a shell command such as "find . -name foo -exec rm {};" but it gives you the ability to be cross platform -- to work anywhere and everywhere. And hey, if you really need to execute a shell command, Ant has an <exec> task that allows different commands to be executed based on the OS that it is executing on.

Pasos para usar ant:

1. genera un archivo válido XML llamado “build.xml”
2. ***corrida normal con ej.bat (sin ant) para ver qué pasos poner***
3. ***adaptar build y correr ant contra “build.xml”***

1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema open source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar para mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la corrección de una clase

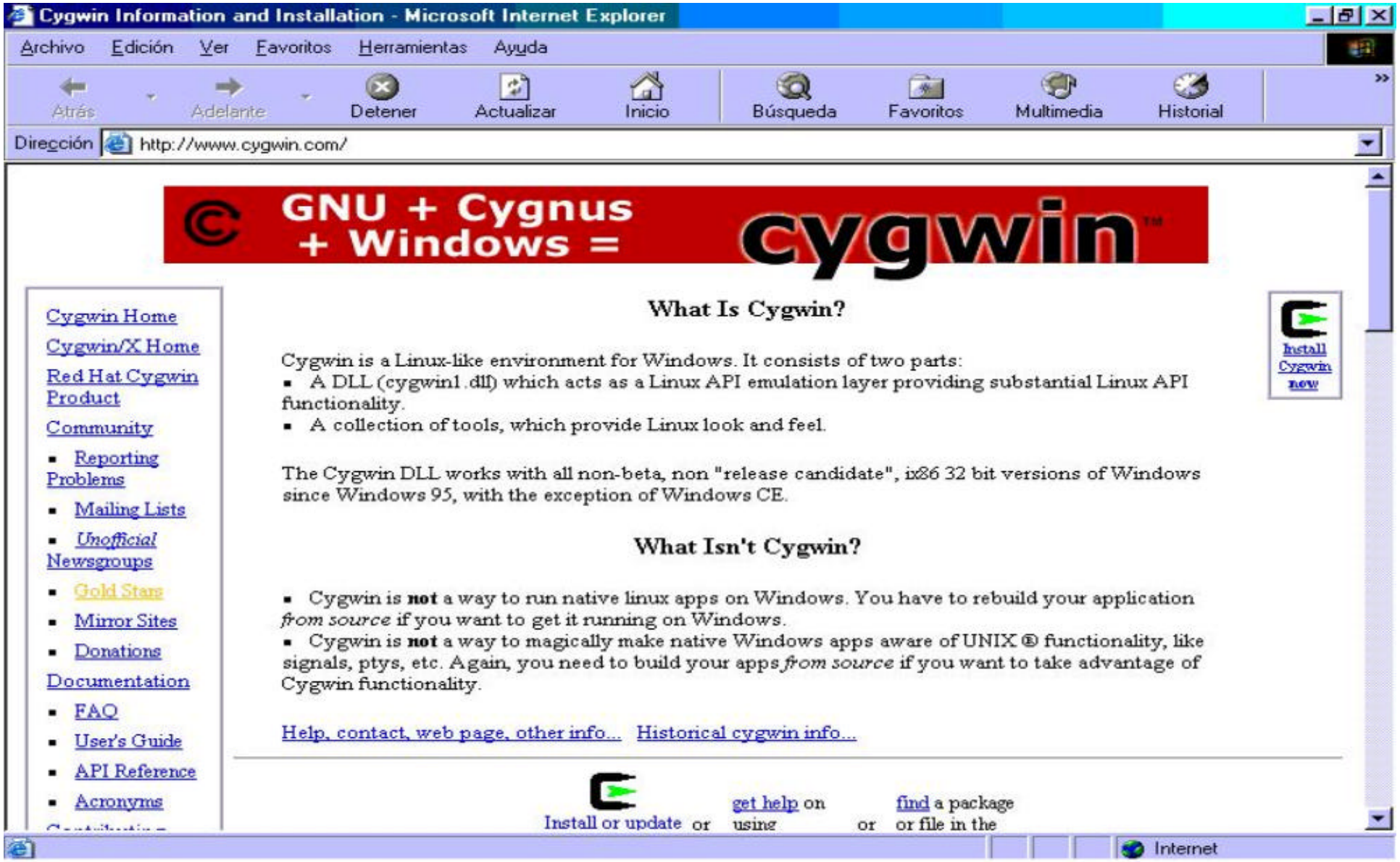
1.b.III. Provistas para Java: CVS
(Concurrent Versions System)
www.cvshome.org

definición:

herramienta estándar para controlar la versión de fuentes en grandes proyectos Java y proveer la distribución de un sistema.

características:

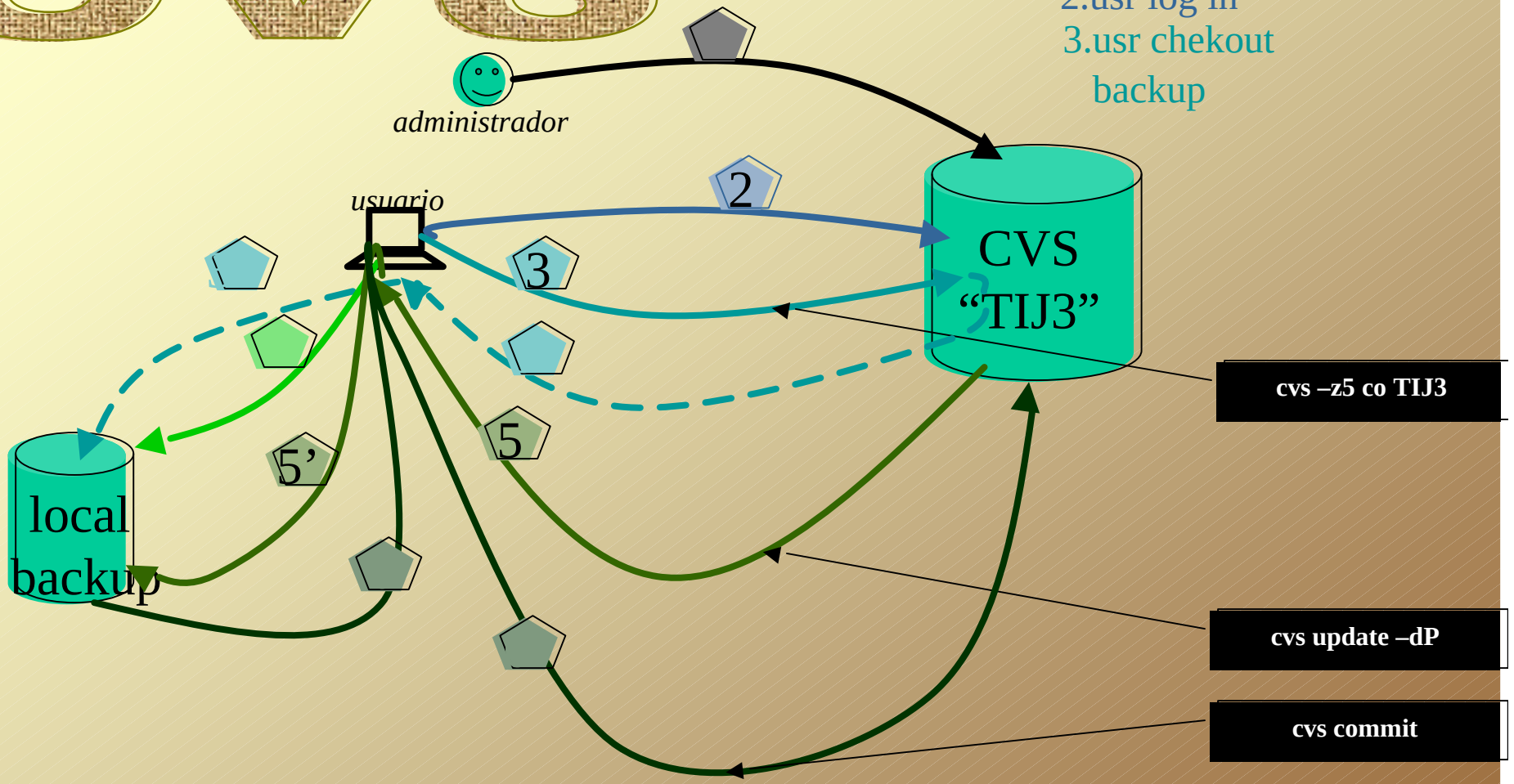
1. necesita de un servidor en internet o en una LAN para alojar los fuentes
2. trabaja sobre la base de user/pwd por lo que :
   - necesita un nivel de seguridad mínimo (en windows se puede trabajar con una versión del protocolo ssh cargando cygwin)



Christopher Faylor

http://cyguin.com

CVS



1.inicializa el sistema
2.usr log in
3.usr chekout
backup

4.usr modification
5.local synchronization
-d: agega directorios
-p: elimina (prune off) directorios locales obsoletos
-resuelve colisiones
6.commit

1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema open source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar para mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la corrección de una clase

1.c.l. Provistas por Java: Assertion

definición:

mecanismo de debug provisto desde JDK 1.4

características:

1. rangos/valores de variables
2. validez de argumentos
3. no se incluye en compilación estándar
4. desactivado por default
5. activable para un package, class o conjunto o global

1.c.1. Provistas por Java: Assertion

6 **Sintaxis**:

assert boolean-expression;
assert boolean-expression: information-expression;

7

activación de Assert

comportamiento Java por default:

- en compilación → resulta el warning

  nnn.java:n: warning: as of release 1.4, assert is a keyword, and may not be used as an identifier

- en ejecución → ignora los assert

para activar cambiando el comportamiento default:

- compilar con flag –source 1.4 → saca warning extra
- ejecutar con flag *–ea* o bien *-enableassertions* → interpreta las assertion

1.Herramientas para PU caja blanca

1.a.Hechas por nosotros

Prueba automática de unidad

cuando hacemos una herramienta para nuestro sistema que permite testear automáticamente un módulo

1.b.Provistas para Java

1.b.I.JUnit

sistema poen source de test estándar de práctica

1.b.II.Ant

herramienta open source estándar para mantener testing y source automáticamente asociados lógicamente

1.b.III.CVS

heramienta open source estándar para mantener un repositorio organizado los source de un proyecto determinado

1.c.Provistas por Java

1.c.I.mecanismo assertion (jdk 1.4)

apoya la verificación de código en runtime

1.c.II.Design By Contract (DBC)

formaliza la correccion de una clase

M. D. López De Luise

1.c.II. Provistas por Java: assert y DBC

DBC=Design By Contract *(Bertrand Meyer)*

Hipótesis

- El comportamiento de un Objeto puede especificarse como si fuera un contrato *(entre el Objeto proveedor del servicio y el cliente consumidor de ese servicio)*
- El comportamiento pactado puede garantizarse si se verifican ciertas condic^{s}.

precondiciones → al entrar al método:verifico argumentos recibidos

DBC *light* 4:
no valido args si necesito performance y es un bottleneck y si es razonablemente segura la fuente de argumentos

postcondiciones → al salir del método:verifico estado al final del método *(antes del return)*

DBC *light* 2:
no postcondición si hago buen PU *(uso assert p/check instruction)*

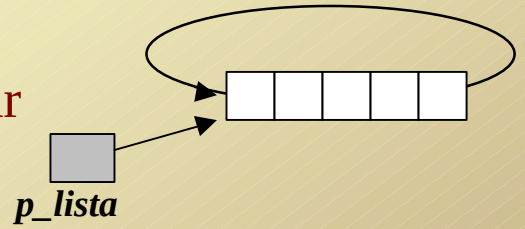
invariantes → después de construir, al entrar y salir del método:verifico el mismo estado del objeto *(no durante ejecución del método)*

DBC *light* 1:
no al entrar al método si lo hago al salir

DBC *light* 3:
no al salir del método si hago buen PU(caja blanca validando estado del objeto)

DBC+PU+assert: Ejemplo de aplicación

1. caso → cola FIFO sobre un vector circular



2. contrato

- precondiciones
  - no se pueden insertar elementos en cola llena
  - no se pueden insertar/obtener elementos null
  - no se pueden insertar elementos en cola null
  - no se pueden obtener elementos de cola vacía
- postcondiciones → no se pueden obtener elementos null
- invariantes
  - la porción de cola FIFO ocupada no puede contener elementos null
  - la porción de cola FIFO desocupada sólo puede contener elementos null

2.Herramientas para PU caja negra

2.a.Provistas por Java

I.logging(jdk 1.4)
reporta información en runtime en reemplazo al uso de println()

II.debbuging (JDB)
mostrar información específica para descubrir problemas

III.profiling (hprof)
analizar el uso de recursos a nivel métodos u objetos

IV.Doclets (javadoc –doclet)
estudiar las clases a partir de la documentación automáticamente generada por javadoc

2.a.I.logging

definición:

proceso para reportar información acerca de un programa en funcionamiento

características:

1. rastreo del estado de un programa implementado
2. alternativa al println para debug

implementación en Java

1. reemplaza el uso de una variable e instrucciones if:

```
....
public static final debug=true;
...
if (debug) {System.out.println("este es un debug");}
```

(desventaja: recompilar cada vez que cambia el valor de *debug)*

2. para usar el mecanismo de logging debo usar clase Logger de la biblioteca:
   java.util.logging.*;

3. el mecanismo se basa en una serie de métodos que permite trabajar con la información de estado de la aplicación.

Recomendaciones generales

1. utilizar un logger por cada clase
   - el nombre del logger debe contener al de la clase
2. como segunda alternativa utilizar un logger por cada package
3. utilizar las técnicas específicas de logging para grandes proyectos
   - configuración por archivos
   - rotación de log files
   - limitación de los respectivos tamaños
   - limitación de cantidad de log files

2.Herramientas para PU caja negra

2.a.Provistas por Java

I.logging API (jdk 1.4)
reporta informacion en runtime en reemplazo al uso de println()

II.debbuging (JDB)
mostrar información específica para descubrir problemas

III.profiling (hprof)
analizar el uso de recursos a nivel métodos u objetos

IV.Doclets (javadoc –doclet)
estudiar las clases a partir de la documentación automáticamente generada por javadoc

2.a.II.debbugging

JDB (Java Debbuger)

1. similitudes con println
   - imprime leyendas
   - imprime valores
2. similitudes con logging
   - imprime leyendas
   - independencia del debug respecto a la codificación
3. algunas diferencias
   - disposición de breakpoints
   - visibilidad del estado del programa en cualquier momento
   - seguimiento línea a línea del código

JDB

definición:

herramienta de tipo consola que viene con jdk para posibilitar el rastreo de errores.

características:

1. es command line debugger
2. es más trabajoso que los debuggers gráficos
3. es gratuito y está disponible con la simple instalación de jdk
4. existe la versión gratuita gráfica:
   - en JCreator (Xinox Software)
   - en Eclipse (IBM)
   - en JBuilder (Borland)

JDB

Pasos:

1. compilar generando la información de debug:
   parámetro -g

2. usar jdb: a través de los comandos de la herramienta se puede
   - establecer breakpoint lógicos (condiciones)
   - establecer breakpoint físicos (líneas de código, nombre de métodos, tipos de excepciones)
   - observar datafields, threads, stack, etc., en cualquier momento
   - alterar el flujo de control del pgm
     - cancelar
     - repetir
     - cambiar de thread
     - suspender un thread
     - continuar con un thread, etc.
   - observar el código fuente en su totalidad o en partes
   - cambiar el valor de data fields

Algunos comandos	Descripción
jdb nomPgm	lanza jdb contra nomPgm
run [clase [args]]	ejecuta el main de esa clase
threads [threadgroup]	lista los threads
suspend [id]	suspende el thread en cuestion
resume [id]	sigue con el thread
where [id|all]	vuelca el stack del thread
print <exp>	imprime el valor de la expresion <exp>
eval <exp>	evalúa una expresión
set <val>=<exp>	asigna el valor de <exp> a <val>
locals	muestra todas las var locales y su valor
classes	muestra todas las clases
class <id>	muestra detalles de la clase <id>
methods <id>	muestra los métodos de la clase <id>
fields <id>	muestra los data fields de la clase <id>
stop in <id>.<method>[(arg_type,...)]	establece un breakpoint en un método
stop at <id>.<line>	establece un breakpoint en una línea
clear <id>.<method>[(arg_type,...)]	borra un breakpoint de un método
clear <id>.<line>	borra un breakpoint de una línea
clear	lista los breakpoints
catch [uncaught|caught|all] <idE>	establece un breakpoint con la excepcion <idE>
watch [access|all] <id>.<field>	muestra acceso/modifs a un campo
unwatch [access|all] <id>.<field>	detiene el comando anterior watch
step	ejecuta la linea actual completa
step up	ejecuta hasta retornar al llamador
stepi	ejecuta el comando actual de la linea
next	avanza una linea (sin entrar en calls)
cont	continua corrida desde el último breakpoint
list [src num|method]	lista el codigo fuente
!!	repite el último comando
help	ayuda

MS-DOS Prompt

corrida con Jdb desde consola

```
D:\DANIEL~1\UPS110~1\LABORA~1\CODIGO\EJEMPLOS\CLASEA~1>javac  -g MiPrueba.java
D:\DANIEL~1\UPS110~1\LABORA~1\CODIGO\EJEMPLOS\CLASEA~1>jdb MiPrueba
Initializing jdb ...
> catch all Error
Deferring all Error.
It will be set after the class is loaded.
> run
run MiPrueba
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: f1()
f2()
f3()

Exception occurred: java.lang.ArithmeticException (uncaught)"thread=main", MiPru
eba.f3(), line=13 |
13          int i = 5 / (--j);

main[1] _
```

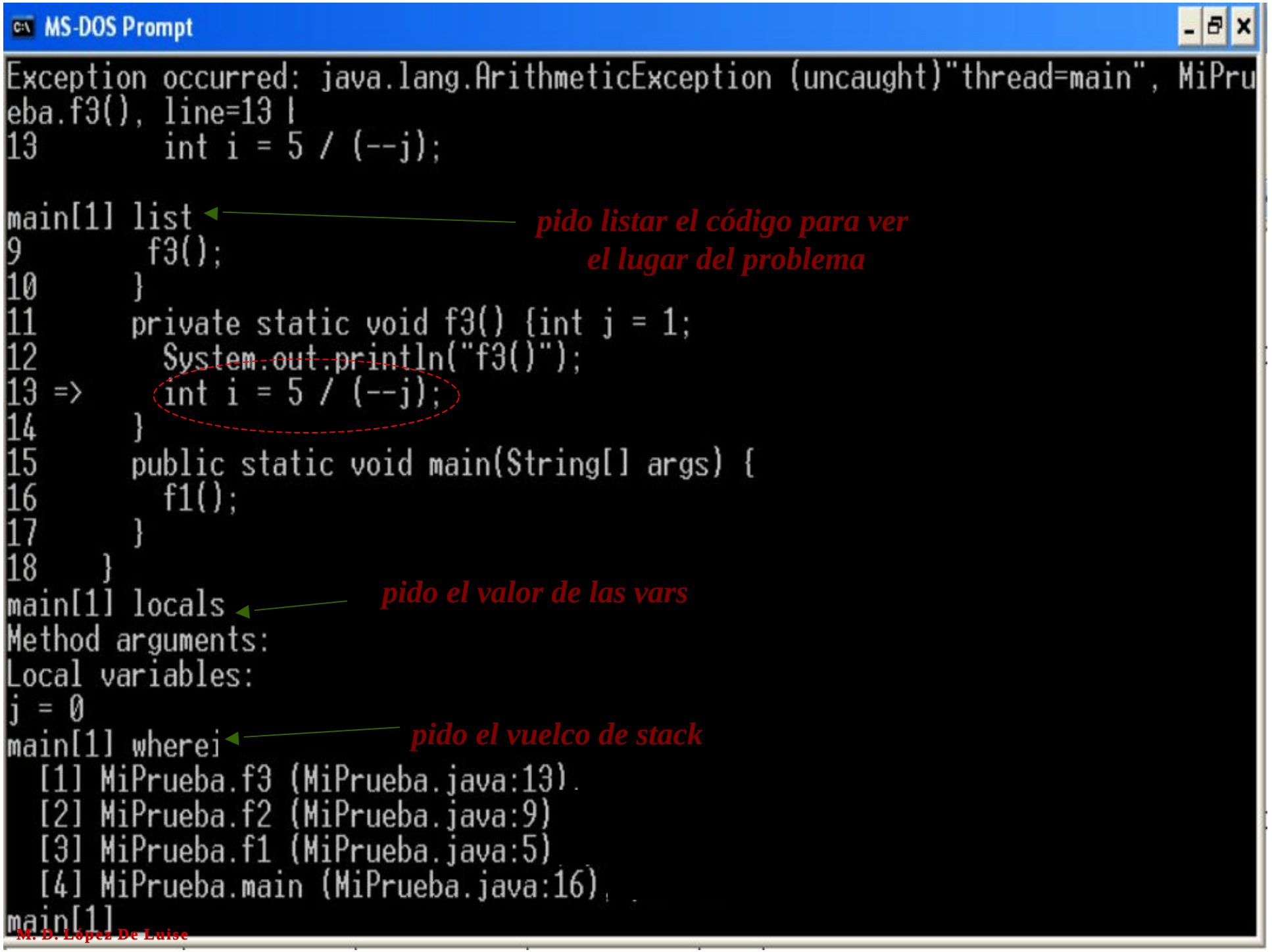
compilo con -g

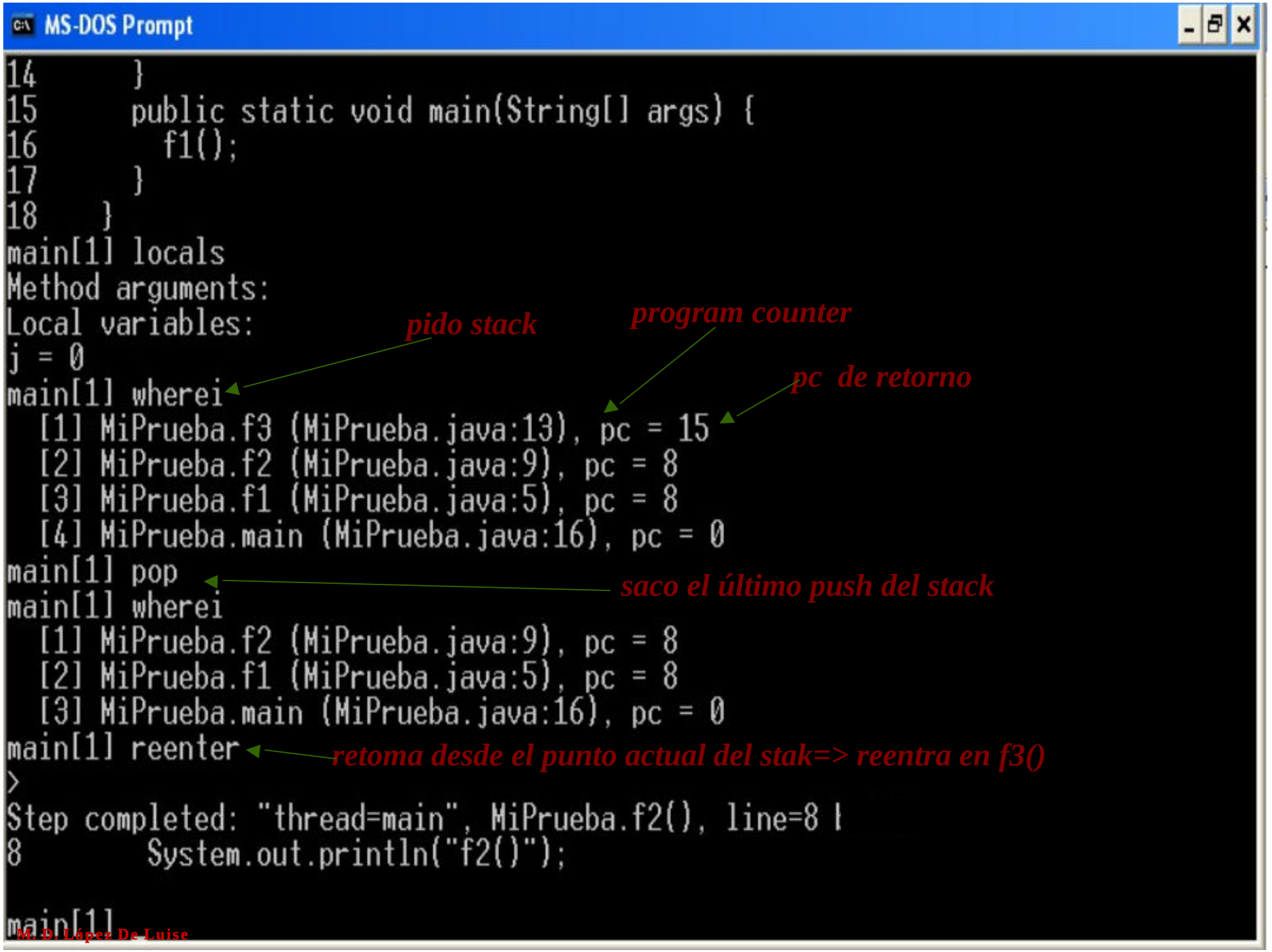
establezco los breakpoint

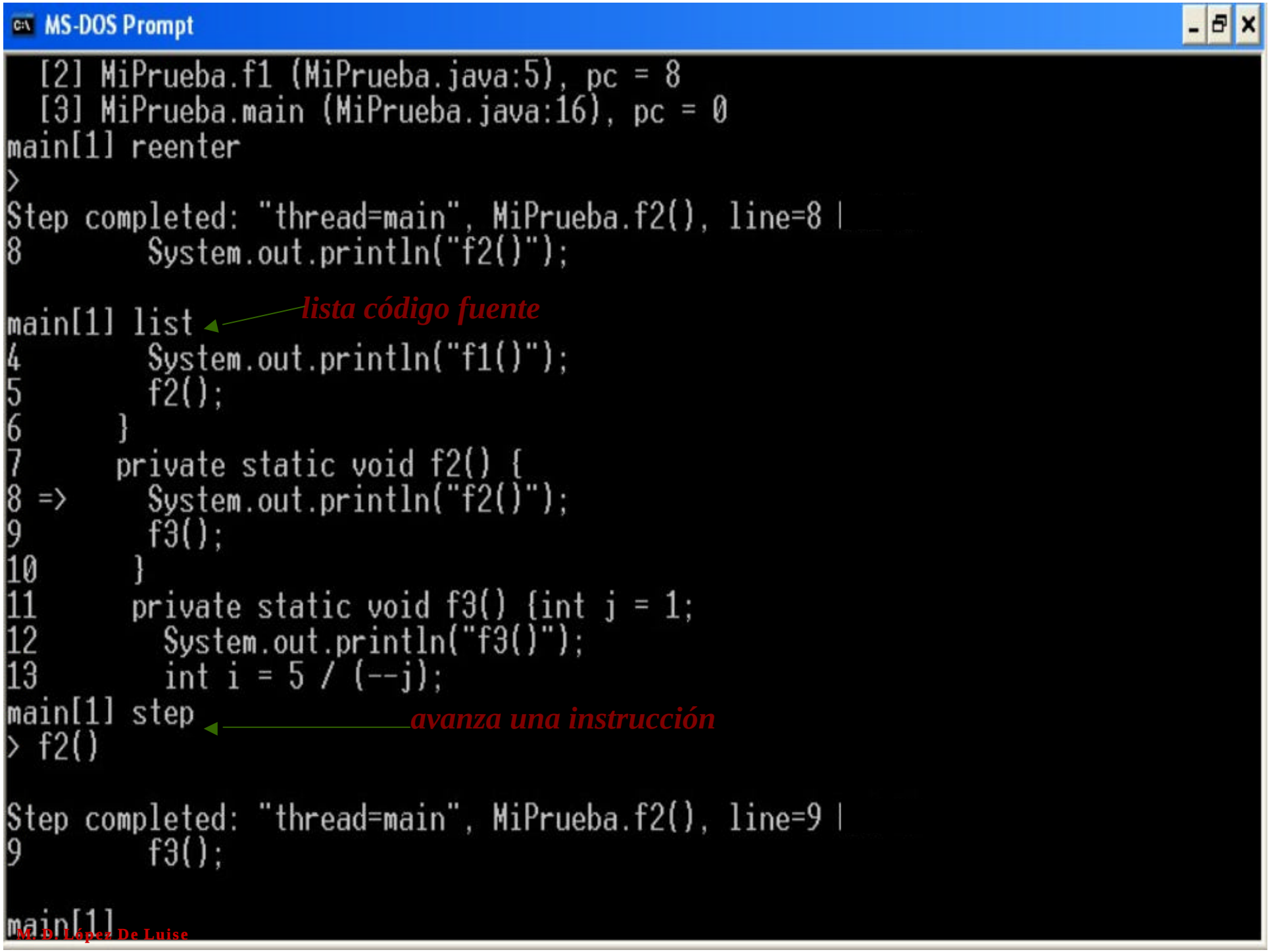
corro jdb contra la clase

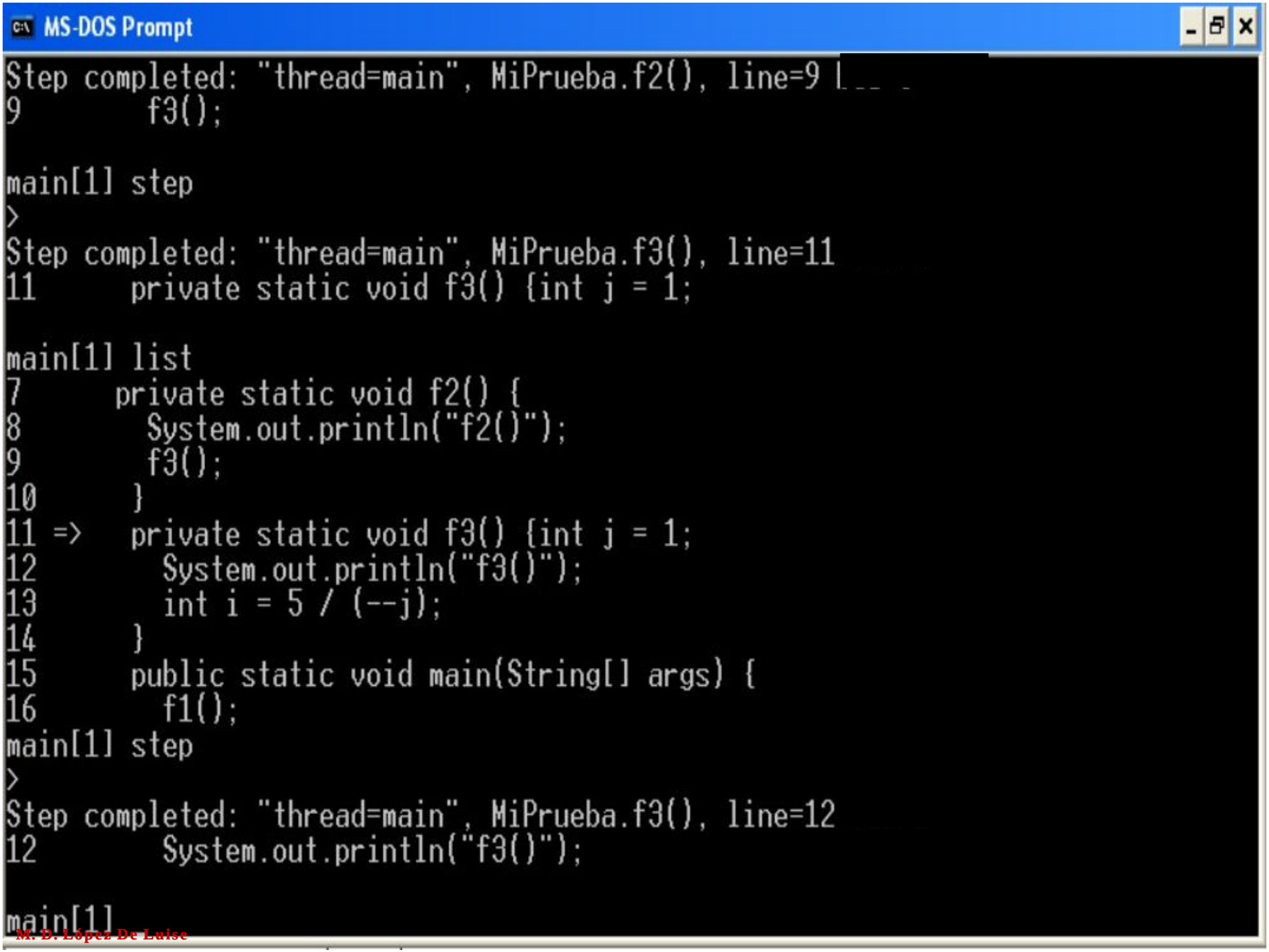
corro la aplicación

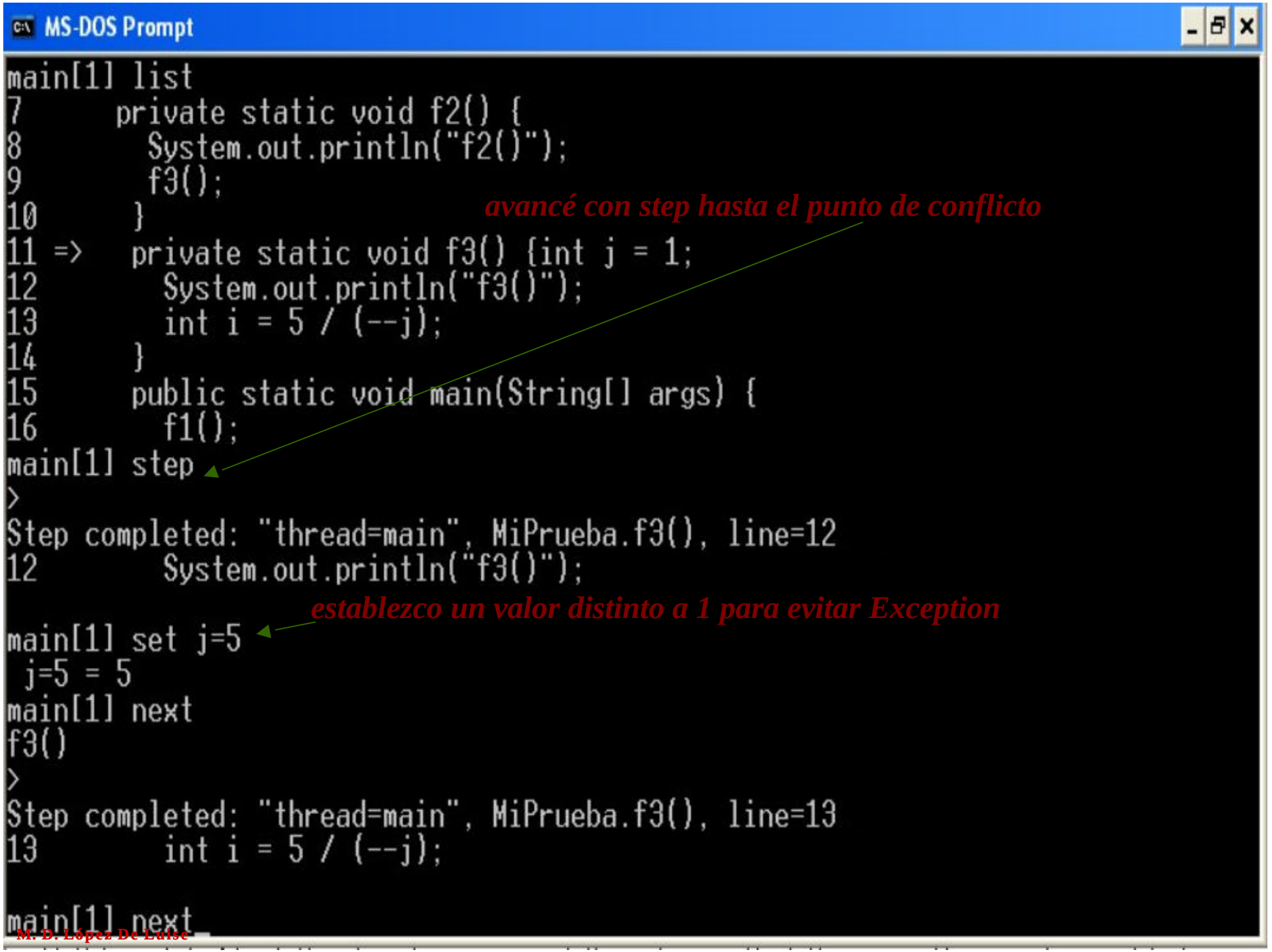
se detiene en la primer Exception

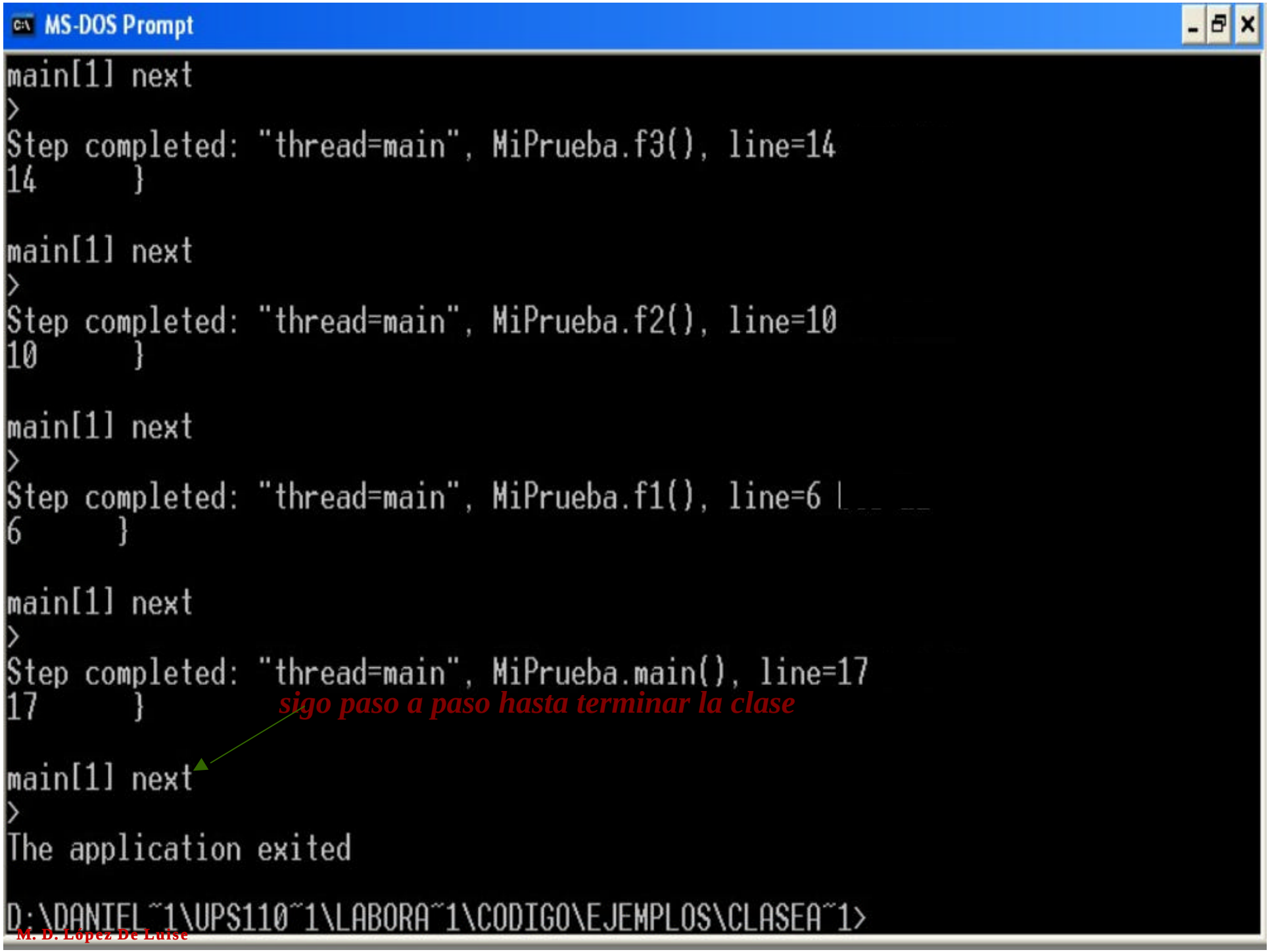

MS-DOS Prompt
Exception occurred: java.lang.ArithmeticException (uncaught)"thread=main", MiPru
eba.f3(), line=13 |
13 int i = 5 / (--j);
main[1] list
9 f3();
10 }
11 private static void f3() {int j = 1;
12 System.out.println("f3()");
13 => int i = 5 / (--j);
14 }
15 public static void main(String[] args) {
16 f1();
17 }
18 }
main[1] locals
Method arguments:
Local variables:
j = 0
main[1] where
[1] MiPrueba.f3 (MiPrueba.java:13)
[2] MiPrueba.f2 (MiPrueba.java:9)
[3] MiPrueba.f1 (MiPrueba.java:5)
[4] MiPrueba.main (MiPrueba.java:16)
main[1]
pido listar el código para ver el lugar del problema
pido el valor de las vars
pido el vuelco de stack


MS-DOS Prompt
14 }
15 public static void main(String[] args) {
16 f1();
17 }
18 }
main[1] locals
Method arguments:
Local variables:
j = 0
main[1] wherei
[1] MiPrueba.f3 (MiPrueba.java:13), pc = 15
[2] MiPrueba.f2 (MiPrueba.java:9), pc = 8
[3] MiPrueba.f1 (MiPrueba.java:5), pc = 8
[4] MiPrueba.main (MiPrueba.java:16), pc = 0
main[1] pop
main[1] wherei
[1] MiPrueba.f2 (MiPrueba.java:9), pc = 8
[2] MiPrueba.f1 (MiPrueba.java:5), pc = 8
[3] MiPrueba.main (MiPrueba.java:16), pc = 0
main[1] reenter
>
Step completed: "thread=main", MiPrueba.f2(), line=8 l
8 System.out.println("f2()");
main[1]
pido stack
program counter
pc de retorno
saco el último push del stack
retoma desde el punto actual del stak=> reentra en f3()


MS-DOS Prompt
[2] MiPrueba.f1 (MiPrueba.java:5), pc = 8
[3] MiPrueba.main (MiPrueba.java:16), pc = 0
main[1] reenter
>
Step completed: "thread=main", MiPrueba.f2(), line=8
8 System.out.println("f2()");
main[1] list
lista código fuente
4 System.out.println("f1()");
5 f2();
6 }
7 private static void f2() {
8 => System.out.println("f2()");
9 f3();
10 }
11 private static void f3() {int j = 1;
12 System.out.println("f3()");
13 int i = 5 / (--j);
main[1] step
avanza una instrucción
> f2()
Step completed: "thread=main", MiPrueba.f2(), line=9
9 f3();
main[1]




MS-DOS Prompt
main[1] list
7 private static void f2() {
8 System.out.println("f2()");
9 f3();
10 }
11 => private static void f3() {int j = 1;
12 System.out.println("f3()");
13 int i = 5 / (--j);
14 }
15 public static void main(String[] args) {
16 f1();
main[1] step
>
Step completed: "thread=main", MiPrueba.f3(), line=12
12 System.out.println("f3()");
main[1] set j=5
j=5 = 5
main[1] next
f3()
>
Step completed: "thread=main", MiPrueba.f3(), line=13
13 int i = 5 / (--j);
main[1] next
avancé con step hasta el punto de conflicto
establezco un valor distinto a 1 para evitar Exception


MS-DOS Prompt
main[1] next
>
Step completed: "thread=main", MiPrueba.f3(), line=14
14 }
main[1] next
>
Step completed: "thread=main", MiPrueba.f2(), line=10
10 }
main[1] next
>
Step completed: "thread=main", MiPrueba.f1(), line=6
6 }
main[1] next
>
Step completed: "thread=main", MiPrueba.main(), line=17
17 }
sigo paso a paso hasta terminar la clase
main[1] next
>
The application exited
D:\DANIEL~1\UPS110~1\LABORA~1\CODIGO\EJEMPLOS\CLASEA~1>

2.Herramientas para PU caja negra

2.a.Provistas por Java

I.logging API (jdk 1.4)
reporta informacion en runtime en reemplazo al uso de println()

II.debbuging (JDB)
mostrar información específica para descubrir problemas

III.profiling (hprof)
analizar el uso de recursos a nivel métodos u objetos

IV.Doclets (javadoc –doclet)
estudiar las clases a partir de la documentación automáticamente generada por javadoc

2.a.III.profiling
HPROF

definición:

herramienta con interface texto que permite analizar el uso de los recursos y optimizar los programas que así lo requieran.

características:

1. debe usarse con precaución para evitar optimización prematura
2. útil para detectar puntos de mayor consumo de memoria/cpu
3. útil para detectar los deadlocks entre threads
   - •análisis de memoria
   - •análisis de uso de CPU
   - •confiabilidad del testing
   - •análisis de threads
4. la actividad de hprof se determina como un argumento de invocación sin espacios
5. toda la información queda almacenada en un archivo java.hprof.txt
6. corre simplemente con el comando:
   - ***java –Xrunjcov:<args> <pgm>***
   - ***java –Xrunhprof:<args><pgm>***

tipo de información que colecta:

1. cantidad de alojamiento de objetos
2. lugares donde se alojaron los objetos
3. métodos de esta clase involucrados en el alojamiento
4. detección de objetos alojados, no usados y no colectados por el gc(***loitering***)
   - abuso de alojamiento de objetos temporarios
   - fallas en la liberación de instancias alojadas dentro de una colección
5. análisis de memoria
6. análisis de uso de CPU
   - cantidad de veces que se invocó un método
   - %de tpo en un método y en sus subordinados
   - Tpo.total neto de procesamiento, espera en IO, locks, etc
7. análisis de threads
8. alcance del testing → líneas no ejecutadas en el testing

corrida de un programa con runjcov

```
Command Prompt

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>javac MiPrueba.java

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>java -Xrunjcov:type=M,exc
lude=java.lang MiPrueba
f1()
f2()
f3()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MiPrueba.f3(MiPrueba.java:17)
        at MiPrueba.f2(MiPrueba.java:9)
        at MiPrueba.f1(MiPrueba.java:5)
        at MiPrueba.main(MiPrueba.java:20)
*** JCOV error : jcov items don't match (line 1006)

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>
```

corrida con HPROF para rastrear

```
Command Prompt

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>java -Xrunhprof:heap=site
s,cpu=samples,thread=y,doe=y MiPrueba
f1()
f2()
f3()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MiPrueba.f3(MiPrueba.java:17)
        at MiPrueba.f2(MiPrueba.java:9)
        at MiPrueba.f1(MiPrueba.java:5)
        at MiPrueba.main(MiPrueba.java:20)
Dumping allocation sites ... CPU usage by sampling running threads ... done.

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>dir *.txt
 Volume in drive D is DLL voyager II
 Volume Serial Number is 70AE-6E52

 Directory of D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1

05/26/2005  18:27             2,020 helpHPROF.txt
05/26/2005  18:52            94,124 java.hprof.txt
05/26/2005  11:28             1,043 src_claseabierta.txt
               3 File(s)         97,187 bytes
               0 Dir(s)   8,120,844,288 bytes free

D:\DANIEL~1\UPS110~1\LABORA~1\codigo\ejemplos\CLASEA~1>
```

- *lugares usados del heap (heap=site)*
- *muestreo estadístico de CPU para ver %uso (cpu=samples)*
- *determinar los thread dentro del stack (thread=y)*
- *hacer vuelco de profiling data al terminar (doe=y)*

archivo generado

Consejos prácticos sacados de la experiencia en estos análisis

1. evitar optimizar a costa de legibilidad
2. balancear optimización Vs esfuerzo de codificación
3. se ha probado que la performance suele ser importante sólo en grandes proyectos
4. suele ser conveniente priorizar la puesta en marcha y luego refinar cíclicamente con el profiling
5. considerar optimización luego del análisis, diseño y deployment salvo que el caso requiera lo contrario
6. no hacer hipótesis y adivinanzas para detectar bottlenecks. Usar profiler.
7. para desactivar un objeto asignar explícitamente null al handler (esto ayuda al gc)
8. considerar la optimización de performance cuando el proyecto es grande, corre por mucho tiempo y es un hito a alcanzar
9. las variables static y final pueden ser muy bien optimizadas por velocidad JVM

2.Herramientas para PU caja negra

2.a.Provistas por Java

I.logging API (jdk 1.4)
reporta informacion en runtime en reemplazo al uso de println()

II.debbuging (JDB)
mostrar información específica para descubrir problemas

III.profiling (hprof)
analizar el uso de recursos a nivel métodos u objetos

IV.Doclets (javadoc –doclet)
estudiar las clases a partir de la documentación automáticamente generada por javadoc

2.a.IV.doclets

definición:

Complemento de la herramienta javadoc, que permite una manipulación personalizada de la información usada por javadoc.

idea:

1. usar el parser de javadoc para estudiar métodos, clases, constructores, accesos, comentarios, etc.
2. usar los doclets para definir handlers que realicen operaciones especiales (cbios. formato, elaboraciones sofisticadas, etc.)
3. facilitar la mejora de código pobre
   - variables no privadas que debieran serlo
   - fallas en el seguimiento de la nomenclatura
   - parámetros indocumentados

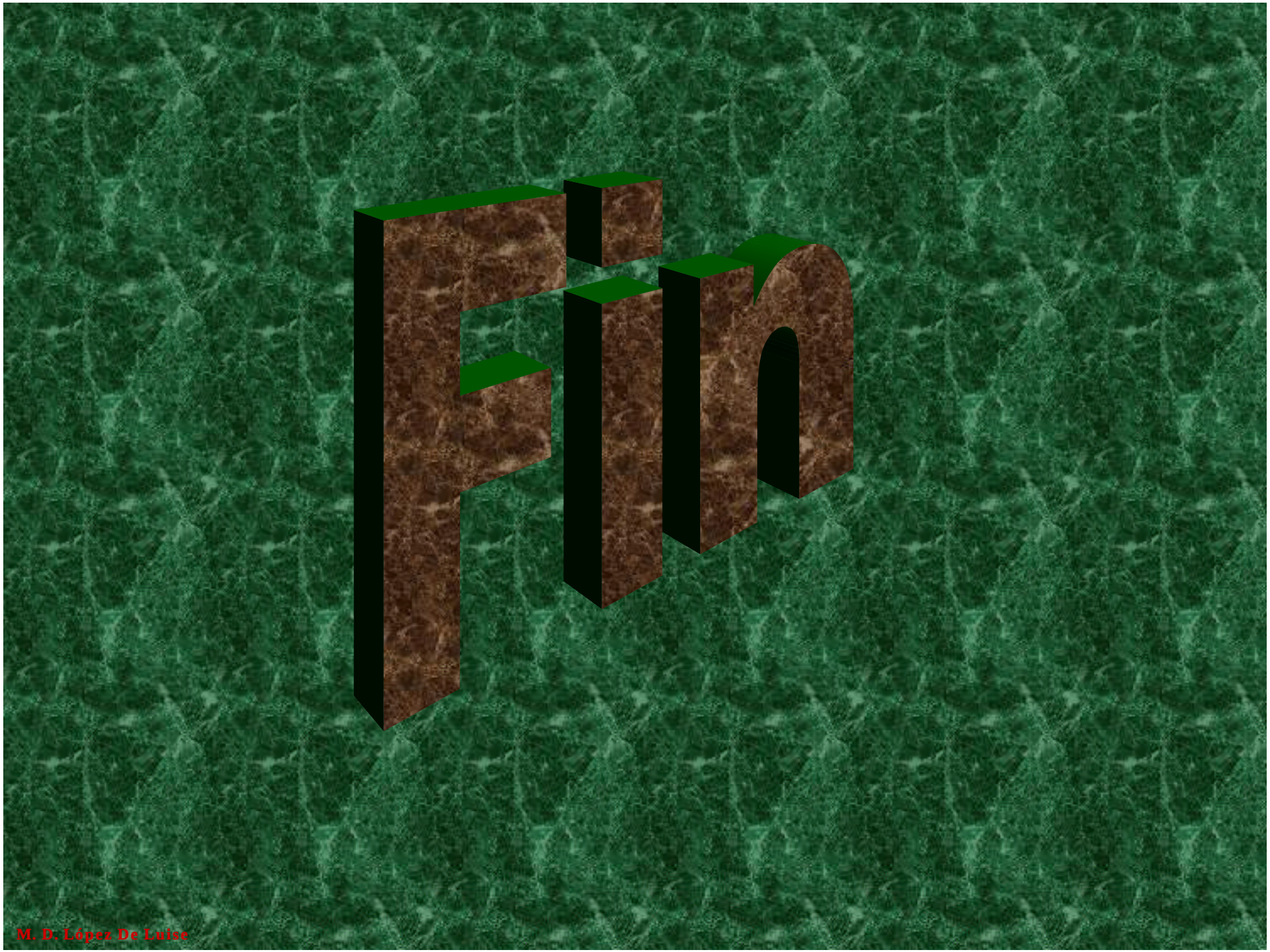
características:

1. se codifica como un programa Java tradicional pero reemplaza el :

   public static void main(String[] args)

   por: ***public static boolean start(RootDoc root)***

2. utiliza la biblioteca com.sun.javadoc


Fin
M. D. López De Luise