Índice

1. Repaso de Java y Eclipse 1

Pr. 1.1. Hola mundo en Java . . . . . . . . . . . . . . . . . . . . . 1

Pr. 1.2. Uso de múltiples \_cheros . . . . . . . . . . . . . . . . . . 2

Pr. 1.3. Creación de un Applet básico . . . . . . . . . . . . . . . . 3

Pr. 1.4. Ciclo de vida de un Applet . . . . . . . . . . . . . . . . . 5

Pr. 1.5. Applet con un botón . . . . . . . . . . . . . . . . . . . . 7

Pr. 1.6. Reacción al evento: clase externa . . . . . . . . . . . . . . 8

Pr. 1.7. Reacción al evento: clase anidada . . . . . . . . . . . . . 9

Pr. 1.8. Reacción al evento: clase anidada no estática . . . . . . . 10

Pr. 1.9. Reacción al evento: clase local . . . . . . . . . . . . . . . 12

Pr. 1.10. Reacción al evento: clases anónimas . . . . . . . . . . . . 15

Pr. 1.11. Reacción al evento: esquema habitual . . . . . . . . . . . 16

Pr. 1.12. Instalación de Eclipse . . . . . . . . . . . . . . . . . . . . 17

Pr. 1.13. Hola mundo en Eclipse . . . . . . . . . . . . . . . . . . . 19

Pr. 1.14. Depuración en Eclipse . . . . . . . . . . . . . . . . . . . . 20

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 24

2. Primeros pasos con Android 25

Pr. 2.1. Instalación de las SDK de Android . . . . . . . . . . . . 26

Pr. 2.2. Instalación de \_plataformas Android\_ . . . . . . . . . . . 27

Pr. 2.3. Creación de Android Virtual Devices . . . . . . . . . . . 28

Pr. 2.4. Explorando el AVD . . . . . . . . . . . . . . . . . . . . . 32

Pr. 2.5. Interacción entre dos AVDs . . . . . . . . . . . . . . . . . 33

Pr. 2.6. La consola de los AVD . . . . . . . . . . . . . . . . . . . 34

Pr. 2.7. Preparación de Eclipse: ADT . . . . . . . . . . . . . . . . 35

Pr. 2.8. El plugin Android Development Tool . . . . . . . . . . . 37

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 38

3. Actividades: las ventanas de Android 39

v

vi Índice

Pr. 3.1. Hola mundo . . . . . . . . . . . . . . . . . . . . . . . . . 39

Pr. 3.2. Nivel de API mínimo y de construcción . . . . . . . . . . 43

Pr. 3.3. Interactuando con el usuario: botón \_Púlsame\_ . . . . . . 44

Pr. 3.4. Ejecución en un dispositivo físico . . . . . . . . . . . . . 46

Pr. 3.5. LogCat: el registro en Android . . . . . . . . . . . . . . . 48

Pr. 3.6. Android Debug Bridge . . . . . . . . . . . . . . . . . . . 49

Pr. 3.7. Depuración de Android . . . . . . . . . . . . . . . . . . . 51

Pr. 3.8. Ciclo de vida de las aplicaciones . . . . . . . . . . . . . . 52

Pr. 3.9. Primera aproximación a los recursos: internacionalización 56

Pr. 3.10. Interfaces de usuario como XML . . . . . . . . . . . . . . 59

Pr. 3.11. Propiedades básicas de los controles . . . . . . . . . . . . 65

Pr. 3.12. Etiquetas . . . . . . . . . . . . . . . . . . . . . . . . . . . 67

Pr. 3.13. Cuadros de texto . . . . . . . . . . . . . . . . . . . . . . 68

Pr. 3.14. Linear Layout . . . . . . . . . . . . . . . . . . . . . . . . 68

Pr. 3.15. ½Adivina mi número! . . . . . . . . . . . . . . . . . . . . . 70

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 74

4. Un poco más sobre UIs 75

Pr. 4.1. RelativeLayout . . . . . . . . . . . . . . . . . . . . . . . . 75

Pr. 4.2. Botones de radio . . . . . . . . . . . . . . . . . . . . . . . 78

Pr. 4.3. Casillas de veri\_cacion . . . . . . . . . . . . . . . . . . . 80

Pr. 4.4. ScrollView . . . . . . . . . . . . . . . . . . . . . . . . . . 81

Pr. 4.5. Mensajes rápidos: toasts . . . . . . . . . . . . . . . . . . 81

Pr. 4.6. Mensajes de alerta con botones . . . . . . . . . . . . . . 83

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 84

5. Intents, \_chero de mani\_esto y componentes 85

Pr. 5.1. Una aplicación con dos ventanas . . . . . . . . . . . . . . 85

Pr. 5.2. AndroidManifest.xml: Actividades . . . . . . . . . . . . . 86

Pr. 5.3. Enviando datos a la segunda ventana . . . . . . . . . . . 90

Pr. 5.4. Recogiendo valores de la segunda ventana . . . . . . . . . 91

Pr. 5.5. Llamando a una actividad de otra aplicación . . . . . . . 93

Pr. 5.6. Invocaciones implícitas con Intents . . . . . . . . . . . . . 96

Pr. 5.7. Permisos . . . . . . . . . . . . . . . . . . . . . . . . . . . 102

Pr. 5.8. Exportación e instalación . . . . . . . . . . . . . . . . . . 103

Pr. 5.9. Receptores de mensajes del sistema . . . . . . . . . . . . 104

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 108

6. Acceso a datos 109

Índice vii

Pr. 6.1. Reconstrucción de una actividad . . . . . . . . . . . . . . 109

Pr. 6.2. Mejorando el juego Adivina mi número . . . . . . . . . . 112

Pr. 6.3. Almacenando preferencias . . . . . . . . . . . . . . . . . 114

Pr. 6.4. Leyendo un \_chero de los recursos . . . . . . . . . . . . . 119

Pr. 6.5. Leyendo y escribiendo en \_cheros externos . . . . . . . . 122

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 128

En el próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . 128

7. Sensores y otro hardware 129

Pr. 7.1. Vibración . . . . . . . . . . . . . . . . . . . . . . . . . . . 129

Pr. 7.2. Sensores . . . . . . . . . . . . . . . . . . . . . . . . . . . 135

Pr. 7.3. Acelerómetro . . . . . . . . . . . . . . . . . . . . . . . . . 137

Pr. 7.4. Grá\_cos en Android . . . . . . . . . . . . . . . . . . . . . 142

Pr. 7.5. Multitouch . . . . . . . . . . . . . . . . . . . . . . . . . . 145

Notas bibliográ\_cas . . . . . . . . . . . . . . . . . . . . . . . . . . . 147

El próximo capítulo . . . . . . . . . . . . . . . . . . . . . . . . . . 148

Capítulo 1

Repaso de Java y Eclipse

Lo último que uno sabe es por dónde

empezar.

Blas Pascal

Resumen: En este capítulo repasaremos algunas de las características

del lenguaje Java que son de importancia para el resto del curso.

Práctica 1.1: Hola mundo en Java

Vamos a crear nuestro primer programa en Java. Abre cualquier editor

de texto, crea un \_chero nuevo llamado HolaMundo.java y escribe:

public clas s HolaMundo {

public s tat ic void main ( St r ing [ ] args ) {

System . out . p r i n t l n ( " ½Hola mundo ! " ) ;

}

} ;

Para compilarlo, necesitamos el JDK (Java Development Kit), en concreto

el compilador de Java, javac:

$ javac HolaMundo.java

Observa la aparición de un nuevo \_chero, HolaMundo.class. Para ejecutar

el código:

$ java HolaMundo

½Hola mundo!

1

2 Capítulo 1. Repaso de Java y Eclipse

Fíjate que ahora hemos usado java (no javac) que es el programa que

contiene la implementación de la JVM. Además, no hemos especi\_cado la

extensión.

\_

Práctica 1.2: Uso de múltiples \_cheros

Crea un nuevo \_chero Fecha.java:

public clas s Fecha {

public void cons t ruye ( int dia , int mes , int anyo ) {

\_dia = dia ;

\_mes = mes ;

\_anyo = anyo ;

}

public void e s c r i b e ( ) {

System . out . p r i n t l n (\_dia + "/" + \_mes + "/" + \_anyo ) ;

}

private int \_dia ;

private int \_mes ;

private int \_anyo ;

}

Crea un nuevo \_chero Main.java, que use el anterior:

public clas s Main {

public s tat ic void main ( St r ing [ ] args ) {

Fecha f = new Fecha ( ) ;

f . cons t ruye (23 , 6 , 1 9 1 2 ) ;

System . out . pr i nt ( "La f e cha e s : " ) ;

f . e s c r i b e ( ) ;

}

}

Compila éste último:

$ javac Main.java

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

3

Observa que se ha compilado automáticamente también la dependencia

y tendrás el \_chero Fecha.class.

Ejecuta el programa recién compilado:

$ java Main

Modi\_ca la clase Fecha.java, eliminando (o comentando) el método

escribe().

Compila esa clase Fecha (½no Main!):

$ javac Fecha.java

Se habrá regenerado el \_chero Fecha.class que ahora no tendrá el

método escribe(), que era usado por Main.java

Prueba otra vez a ejecutar el programa:

$ java Main

\_

Práctica 1.3: Creación de un Applet básico

Crea un nuevo \_chero Java, AppletBasico.java, con el siguiente contenido:

import java . appl e t . Applet ;

public clas s AppletBas ico extends Applet {

public void i n i t ( ) {

add (new java . awt . Label ( "Hola mundo" ) ) ;

}

}

Compila el \_chero:

$ javac AppletBasico

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

4 Capítulo 1. Repaso de Java y Eclipse

Si lo intentas ejecutar, recibirás un error porque no tenemos método

main.

Un applet está pensado para ser ejecutado incrustado en una página

Web. Crea el \_chero AppletBasico.html con el contenido que aparece

a continuación:

<!DOCTYPE html>

<html>

<head>

<t i t l e>Mi appl e t</ t i t l e>

</head>

<body>

<h1>Mi appl e t</h1>

<applet code="AppletBas ico . c l a s s "

width=" 300 "

height=" 200 ">

</applet>

</body>

</html>

Abre la página .html con un navegador. Si tiene habilitado Java, verás

el applet:

También puedes usar un visor de applets que proporciona el JDK. Le

tenemos que especi\_car el .html, aunque luego únicamente mostrará el

applet:

$ appletviewer AppletBasico.html

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

5

\_

Práctica 1.4: Ciclo de vida de un Applet

Un applet no tiene método main. En su lugar se ejecuta en el contexto

del navegador, que crea un espacio para él, y llama a sus métodos cuando

se necesita. Se dice que los applets tienen un ciclo de vida controlado por

el contenedor. Los métodos con los que el applet recibe noti\_caciones de

cambios en su ciclo de vida son:

init(): llamado cuando el applet acaba de ser cargado por el contenedor.

Es invocado una única vez, antes de la primera llamada a start().

Se utiliza para realizar inicializaciones básicas, como preparar el interfaz

grá\_co, cargar recursos, crear hebras, etcétera.

start(): se llama para indicar que la ejecución comienza. Es invocado

tras init().

stop(): se llama para indicar que la ejecución se detiene (al menos

temporalmente). Ocurre cuando se minimiza la ventana.

destroy(): opuesto a init(), es invocado una única vez al \_nal de la

vida del applet, para pedirle que libere todos sus recursos antes de que

el contenedor lo elimine de\_nitivamente. Normalmente se realizarán las

acciones inversas a las que se realizaron en init(), liberando todos los

recursos.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

6 Capítulo 1. Repaso de Java y Eclipse

Los métodos start() y stop() son simétricos, y se utilizan típicamente

para iniciar o detener efectos visuales o de sonido, de manera que el applet

no consuma recursos cuando esté minimizado.

La implementación de la clase Applet prede\_nida tiene esos métodos

vacíos. Los applets concretos que se programan (que deben heredar de dicha

clase) sobreescribirán los métodos que les interesen, añadiendo el código que

necesiten.

Para probarlo, vamos a hacer un applet que escribirá por la salida estándar

una cadena indicando que se ha invocado a cada uno de los métodos.

Ten en cuenta que esa salida estándar normalmente no se verá si el applet

se ejecuta directamente en el navegador.

Crea un \_chero nuevo, CicloVidaApplet.java con el contenido:

import java . appl e t . Applet ;

public clas s CicloVidaApplet extends Applet {

public void i n i t ( ) {

System . out . p r i n t l n ( " i n i t " ) ;

}

public void s t a r t ( ) {

System . out . p r i n t l n ( " s t a r t " ) ;

}

public void s top ( ) {

System . out . p r i n t l n ( " s top " ) ;

}

public void de s t r oy ( ) {

System . out . p r i n t l n ( " de s t r oy " ) ;

}

}

Compila el código que acabas de crear.

javac CicloVidaApplet.java

Crea el .html contenedor del applet de una manera semejante al de la

práctica 1.3.

Lanza con appletviewer el applet y observa la consola. Verás aparecer

el texto init y start.

Minimiza y maximiza la ventana del applet varias veces. Comprueba

que aparecen stop y start adecuadamente.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

7

Cierra el visor, y comprueba que aparece destroy.

La aplicación appletviewer proporciona un menú con el que podemos

controlar el ciclo de vida para realizar pruebas durante el desarrollo. Experimenta

con sus opciones.

\_

Práctica 1.5: Applet con un botón

El JDK proporciona numerosos \_controles\_ (o widgets). Ya hemos visto

un tipo, las etiquetas, que se consiguen gracias a la clase java.awt.Label.

En esta práctica vamos a sustituir la etiqueta por un botón.

Crea un nuevo \_chero, AppletConBoton.java y escribe el siguiente

código:

import java . appl e t . Applet ;

public clas s AppletConBoton extends Applet {

public void i n i t ( ) {

java . awt . Button boton ;

boton = new java . awt . Button ( "Pulsame" ) ;

add ( boton ) ;

} // i n i t

} // AppletConBoton

En este caso, la creación del botón e inclusión en el applet la hemos

separado en tres instrucciones. Podríamos habernos ahorrado código

resumiendo todo el código de init() con:

add(new java.awt.Button("Pulsame"));

Lo hemos separado en tres líneas porque más adelante haremos uso del

botón.

Compila la clase, crea el \_chero .html correspondiente, y lánzalo con

appletviewer.

Observa que la ventana muestra ahora un botón que reacciona al ratón,

aunque el programa no hace nada cuando se le pulsa.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

8 Capítulo 1. Repaso de Java y Eclipse

\_

Práctica 1.6: Reacción al evento: clase externa

Las instancias de la clase java.awt.Button noti\_can el evento de pulsaci

ón a los oyentes que se hayan registrado en ellas. En concreto, tendrán que

implementar el interfaz java.awt.event.ActionListener, y de\_nir el mé-

todo public void actionPerformed(java.awt.event.ActionEvent e).

Para eso, podemos crear un \_chero nuevo MiActionListener.java que

implemente esa clase, y luego utilizarla desde la clase del applet. Para evitar

la proliferación de muchos \_cheros de código fuente, sin embargo, Java nos

permite de\_nir en un mismo \_chero más de una clase, aunque sólo una podrá

ser pública.

Crea un nuevo \_chero llamado BotonClaseExterna.java y escribe el

siguiente código:

import java . appl e t . Applet ;

// Clase que manejará e l e v ento de pu l s a c i ón d e l botón .

// Observa que e s t a c l a s e no es p ú b l i c a (no comienza con

// p u b l i c ) .

clas s MiAc t ionLi s t ene r

implements java . awt . event . Ac t i onLi s t ene r {

public void ac t ionPe r formed ( java . awt . event . ActionEvent e ) {

System . out . p r i n t l n ( " ½Pul sado ! " ) ;

} // act ionPer formed

} // c l a s s MiAct ionLi s tener

public clas s BotonClaseExterna extends Applet {

public void i n i t ( ) {

java . awt . Button boton ;

boton = new java . awt . Button ( "Pulsame" ) ;

// Línea nueva

boton . addAc t ionLi s t ene r (new MiAc t ionLi s t ene r ( ) ) ;

add ( boton ) ;

} // i n i t

} // c l a s s BotonClaseExterna

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

9

Compila la clase. Observa que aparecen dos \_cheros .class, uno para

cada una de las clases.

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa el botón, y comprueba cómo aparece en la consola el texto

½Pulsado!.

\_

Práctica 1.7: Reacción al evento: clase anidada

Con la solución anterior, la clase oyente está \_al mismo nivel\_ que la del

applet, cosa que conceptualmente no es demasiado limpio. Además, podríamos

querer acceder a atributos estáticos de la clase del applet desde el oyente,

y no lo podremos hacer a no ser que sean públicos.

Para solucionar las dos cosas, Java permite la de\_nición anidada de clases.

En inglés se conoce como top-level nested classes and interfaces. A nosotros

sólo nos interesan aquí las clases.

Crea un nuevo \_chero llamado BotonClaseAnidada.java y escribe el

siguiente código:

import java . appl e t . Applet ;

public clas s BotonClaseAnidada extends Applet {

//􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀

// Clase anidada , dent ro de ot ra . Observa e l s t a t i c .

s tat ic clas s MiAc t ionLi s t ene r

implements java . awt . event . Ac t i onLi s t ene r {

public void ac t ionPe r formed ( java . awt . event . ActionEvent e ) {

// Accedemos a l a t r i b u t o p r o t e g i d o de l a c l a s e a l a

// que pertenecemos .

System . out . p r i n t l n (mensaje ) ;

}

} // c l a s s MiAct ionLi s tener

//􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀

public void i n i t ( ) {

java . awt . Button miBoton = new java . awt . Button ( "Pulsame" ) ;

miBoton . addAc t ionLi s t ene r (new MiAc t ionLi s t ene r ( ) ) ;

add (miBoton ) ;

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

10 Capítulo 1. Repaso de Java y Eclipse

} // i n i t

// Mensaje que queremos mostrar cuando se p u l s e e l botón .

protected s tat ic St r ing mensaje="½AY! " ;

} // c l a s s BotonClaseAnidada

Las cosas a tener en cuenta son:

\_ La clase MiActionListener se encuentra ahora dentro de la clase

del applet.

\_ La clase del applet tiene un nuevo atributo estático protegido

mensaje con el texto que se quiere mostrar cuando se pulsa el

botón.

\_ La clase está de\_nida como static.

\_ El método actionPerformed de la clase interna accede al atributo

estático, aunque sea protegido.

Compila la clase. Observa que aparecen dos \_cheros .class, pero ahora

el del listener tiene un nombre cuali\_cado con el nombre de la clase a

la que pertenece (BotonClaseAnidada$MiActionListener.class).

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa el botón, y comprueba cómo aparece en la consola el texto

½Pulsado!.

\_

Práctica 1.8: Reacción al evento: clase anidada no estática

La siguiente evolución es que la clase anidada no sea estática (no tenga

el modi\_cador static). En ese caso, la clase interna podrá acceder no sólo

a los atributos estáticos, sino también a los dinámicos. Aquí se dispara la

pregunta ¾de qué objeto?

La respuesta es sencilla. Las clases aninadas no estáticas (non-static inner

classes) tienen una restricción que no tenían sus versiones estáticas: siempre

deben crearse en el contexto de un objeto de la clase externa. De hecho, no

se pueden crear instancias suyas fuera de la clase; sólo podrán crearse en

métodos de instancia de dicha clase externa. Con esta restricción, el objeto

al que pertenecen los atributos accedidos resulta natural: aquél que creó el

objeto.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

11

La posibilidad de acceder a atributos dinámicos no es especialmente útil

en el ejemplo que estamos utilizando. Sólo lo sería si quisiéramos tener varias

instancias de la clase externa, cada una con valores diferentes en el atributo

accedido desde la clase interna. Dado que la clase externa es el applet, y sólo

vamos a tener una instancia, resulta complicado poner un ejemplo sencillo

donde sea de utilidad. En cualquier caso, podemos verlo funcionando:

Crea un nuevo \_chero llamado BotonClaseAnidadaNoEstatica.java

y escribe el siguiente código:

import java . appl e t . Applet ;

public clas s BotonClaseAnidadaNoEstatica extends Applet {

//􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀

// Clase anidada , dent ro de ot ra . Observa que ahora no t i e n e

// e l s t a t i c .

clas s MiAc t ionLi s t ene r

implements java . awt . event . Ac t i onLi s t ene r {

public void ac t ionPe r formed ( java . awt . event . ActionEvent e ) {

// Accedemos a l a t r i b u t o dinámico (no e s t á t i c o )

// p r o t e g i d o de l a c l a s e a l a que pertenecemos .

System . out . p r i n t l n (mensaje ) ;

}

} // c l a s s MiAct ionLi s tener

//􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀􀀀

public void i n i t ( ) {

java . awt . Button miBoton = new java . awt . Button ( "Pulsame" ) ;

miBoton . addAc t ionLi s t ene r (new MiAc t ionLi s t ene r ( ) ) ;

add (miBoton ) ;

}

// Mensaje que queremos mostrar cuando se p u l s e e l botón .

protected St r ing mensaje="½AY! " ;

} // c l a s s BotonClaseAnidadaNoEstat ica

Las cosas a tener en cuenta son:

\_ Ahora la clase MiActionListener no está etiquetada como static.

\_ El atributo mensaje sigue siendo protegido, pero ahora es diná-

mico en lugar de estático.

\_ El método actionPerformed de la clase interna accede al atributo

no estático, incluso aunque sea protegido.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

12 Capítulo 1. Repaso de Java y Eclipse

Compila la clase. Observa que aparecen dos \_cheros .class, igual que

antes.

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa el botón, y comprueba cómo aparece en la consola el texto

½Pulsado!.

\_

Práctica 1.9: Reacción al evento: clase local

El paso siguiente es de\_nir la clase dentro de un bloque de código. En

principio, es similar a las clases internas no estáticas anteriores, salvo que en

lugar de de\_nirlo a nivel de la clase externa, se de\_ne, por ejemplo, dentro de

un método. Ahora la clase no podrá ser visible fuera nunca (serán privadas al

bloque), por lo que no necesita especi\_car su visibilidad, no podrán crearse

instancias suyas fuera del bloque, y no tiene sentido que tengan miembros

estáticos.

Crea un nuevo \_chero llamado BotonClaseLocal1.java y escribe el

siguiente código:

import java . appl e t . Applet ;

public clas s BotonClaseLocal1 extends Applet {

public void i n i t ( ) {

// Clase l o c a l , pr i vada d e l método i n i t ( ) . No s e rá

// v i s i b l e fue r a .

clas s MiAc t ionLi s t ene r

implements java . awt . event . Ac t i onLi s t ene r {

public void ac t ionPe r formed (

java . awt . event . ActionEvent e ) {

// Accedemos a l a t r i b u t o dinámico (no e s t á t i c o )

// p r o t e g i d o de l a c l a s e a l a que pertenecemos .

System . out . p r i n t l n (mensaje ) ;

}

} // c l a s s MiAct ionLi s tener

java . awt . Button miBoton = new java . awt . Button ( "Pulsame" ) ;

miBoton . addAc t ionLi s t ene r (new MiAc t ionLi s t ene r ( ) ) ;

add (miBoton ) ;

} // i n i t

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

13

// Mensaje que queremos mostrar cuando se p u l s e e l botón .

protected St r ing mensaje="½AY! " ;

} // BotonClaseLocal1

La diferencia principal con la versión anterior es que ahora la clase

MiActionListener está dentro del método init() de la clase externa.

Compila la clase. Observa que aparecen dos \_cheros .class. Ahora

el nombre de la clase local es algo diferente, con un 1 de separación

(BotonClaseLocal1$1MiActionListener.class). El número se necesita

porque podríamos de\_nir dentro de métodos diferentes clases locales

con el mismo nombre, y es necesario un mecanismo que evite las

colisiones de nombres entre los .class. El número se iría incrementando

con la repetición del nombre de la clase.

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa el botón, y comprueba cómo aparece en la consola el texto

½Pulsado!.

En principio, parece que las clases locales no aportan nada respecto a las

clases internas no estáticas anteriores. La diferencia que las hace especiales es

que pueden acceder a los atributos y variables locales del método donde se han

de\_nido y creado, siempre que éstos sean final (constantes). Esto supone

un acercamiento a \_los cierres\_ (closures) de la programación funcional. Es

decir, si el método init() anterior tuviera una variable local (final), el

método actionPerformed() de la clase local podría acceder a él.

Esto es útil especialmente con los parámetros. Veamos un ejemplo que lo

demuestra. Para eso, el botón lo crearemos en un método nuevo que recibirá

parámetros, y lo invocaremos varias veces para poner varios botones en el

applet.

Crea un nuevo \_chero llamado BotonClaseLocal2.java y escribe el

siguiente código:

import java . appl e t . Applet ;

public clas s BotonClaseLocal2 extends Applet {

// Método que añade a l a p p l e t un botón con l a e t i q u e t a

// d e l primer parámetro , y que , a l s e r pul sado ,

// e s c r i b i r á en l a s a l i d a e s tándar e l segundo parámetro .

public void ponBoton ( St r ing texto , f inal St r ing pul sado ) {

// Clase l o c a l a l método con e l a c t i on l i s t e n e r .

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

14 Capítulo 1. Repaso de Java y Eclipse

clas s MiAc t ionLi s t ene r

implements java . awt . event . Ac t i onLi s t ene r {

public void ac t ionPe r formed (

java . awt . event . ActionEvent e ) {

// Accedemos tanto a l a t r i b u t o de l a c l a s e

// externa , como a l parámetro ( g r a c i a s a que es

// f i n a l ) .

System . out . p r i n t l n (mensaje + " : " + pul sado ) ;

} // act ionPer formed

} // c l a s s MiAct ionLi s tener

java . awt . Button miBoton = new java . awt . Button ( t ext o ) ;

miBoton . addAc t ionLi s t ene r (new MiAc t ionLi s t ene r ( ) ) ;

add (miBoton ) ;

} // ponBoton

public void i n i t ( ) {

// Ponemos t r e s botones , cada uno con un t e x t o

// y un mensaje .

ponBoton ( "Uno" , "1" ) ;

ponBoton ( "Dos" , "2" ) ;

ponBoton ( "Tres " , "3" ) ;

} // i n i t

protected St r ing mensaje="½AY! " ;

} // c l a s s BotonClaseLocal2

Las cosas a tener en cuenta son:

\_ Se ha añadido un método ponBoton() en el applet que añade un

botón. El método es llamado tres veces en init(), por lo que el

applet tendrá ahora tres botones.

\_ El método ponBoton() recibe dos parámetros. El primero indica el

texto de la etiqueta del botón. El segundo indica lo que queremos

que el botón muestre por la salida estándar al ser pulsado. Observa

que ese atributo es final (constante).

\_ El método contiene la clase local MiActionListener que en el

método actionPerformed accede al parámetro del método que

lo contiene. ½Esto es una cosa muy peculiar! Cuando el método

actionPerformed() sea llamado (el usuario haga click), en realidad

la ejecución del método ponBoton() ½habrá terminado hace

tiempo!

Compila la clase. Observa que aparecen dos \_cheros .class, y el nombre

de la clase local sigue el mismo esquema que antes.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

15

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa los botones, y observa como el mensaje varía.

\_

Práctica 1.10: Reacción al evento: clases anónimas

La última vuelta de tuerca es el uso de clases sin nombre. En el ejemplo

anterior estamos creando una clase llamada MiActionListener local al mé-

todo ponBoton() de la que luego únicamente creamos una instancia y no la

usamos más. Además, de esa clase, lo único que nos interesa es sobreescribir

un método.

Para abreviar la escritura de código, Java se amplió para soportar las

llamadas clases anónimas. Nos permite así hacer a la vez la creación de

la instancia (que pasamos en la invocación a addActionListener()) y la

de\_nición de la clase (que no tendrá nombre). El código es un poco confuso en

principio, porque aparecen muchos \_signos de puntuación\_ de cierre seguidos,

aunque termina resultando natural una vez que nos acostumbramos a esta

construcción.

Crea un nuevo \_chero llamado BotonClaseAnonima.java y escribe el

siguiente código:

import java . appl e t . Applet ;

public clas s BotonClaseAnonima extends Applet {

// Método que añade a l a p p l e t un botón con l a

// e t i q u e t a d e l primer parámetro , y que , a l s e r

// pul sado , e s c r i b i r á en l a s a l i d a e s tándar e l

// segundo parámetro .

public void ponBoton ( St r ing texto ,

f inal St r ing pul sado ) {

java . awt . Button miBoton = new java . awt . Button ( t ext o ) ;

// En l a l lamada a addAc t ionLi s t ene r hacemos e l

// new y de f inimos l a c l a s e di r e c tament e . Por

// nombre de l a c l a s e ponemos e l nombre de l a

// " s u p e r c l a s e " ( i n t e r f a z ) que vamos a

// implementar , lue g o e l método , y cerramos

// todo .

miBoton . addAc t ionLi s t ene r (

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

16 Capítulo 1. Repaso de Java y Eclipse

new java . awt . event . Ac t i onLi s t ene r ( ) {

public void ac t ionPe r formed (

java . awt . event . ActionEvent e ) {

System . out . p r i n t l n (mensaje + " : " +

pul sado ) ;

} // act ionPer formed

} ) ; // Cerramos l a d e f i n i c i ó n de l a c l a s e

// y d e l método

add (miBoton ) ;

} // ponBoton

public void i n i t ( ) {

// Ponemos t r e s botones , cada uno con un t e x t o

// y un mensaje .

ponBoton ( "Uno" , "1" ) ;

ponBoton ( "Dos" , "2" ) ;

ponBoton ( "Tres " , "3" ) ;

} // i n i t

protected St r ing mensaje="½AY! " ;

} // c l a s s BotonClaseAnonima

Las cosas a tener en cuenta son:

\_ Ahora la clase MiActionListener ya no existe. En su lugar, la

invocación a addActionListener() es mucho más complicada.

\_ En la invocación hacemos el new de una clase anónima. En lugar

del nombre de la clase, aparece su de\_nición, indicando la superclase

(o el interfaz) que implementa, y de\_niendo los métodos que

sobreescribe o proporciona.

\_ Desde la clase anónima podemos seguir accediendo a los atributos

del objeto que la ha creado, y también a los parámetros y variables

locales \_nales del método donde se de\_ne.

Compila la clase. Observa que aparecen dos \_cheros .class. Como la

clase anónima no tiene nombre, se utiliza BotonClaseAnonima$1.class

como nombre del \_chero.

Crea el \_chero .html correspondiente, y lánzalo con appletviewer.

Pulsa los botones, y observa como el mensaje varía.

\_

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

17

Práctica 1.11: Reacción al evento: esquema habitual

En las prácticas anteriores hemos querido probar las características que

proporciona Java, y que fueron añadidas principalmente para la programaci

ón de interfaces grá\_cas de usuario y la captura de eventos. No obstante,

los últimos usos que hemos hecho, con el uso del parámetro final, no es

demasiado habitual. En realidad, la construcción más normal es aquella en

la que la clase (en nuestro caso el applet) tiene un método para procesar el

evento, que es invocado gracias a una clase anónima.

Puedes ver la idea en el código siguiente:

import java . appl e t . Applet ;

public clas s BotonFinal extends Applet {

public void i n i t ( ) {

java . awt . Button miBoton = new java . awt . Button ( "Pulsame" ) ;

// Creación de l a c l a s e anónima y e l procesado d e l

// e v ento . Llamamos a l método de l a c l a s e e x t e rna que

// hará e l t r a b a j o .

miBoton . addAc t ionLi s t ene r (

new java . awt . event . Ac t i onLi s t ene r ( ) {

public void ac t ionPe r formed (

java . awt . event . ActionEvent e ) {

botonPulsado ( ) ;

}

} ) ; // Ci e r r e de l a c l a s e y l a l lamada a l método

add (miBoton ) ;

} // i n i t

// Método l lamado cuando se pul s a e l botón d e l a p p l e t .

protected void botonPulsado ( ) {

System . out . p r i n t l n (mensaje ) ;

} // botonPul sado

// Mensaje que queremos mostrar cuando se p u l s e e l botón .

protected St r ing mensaje="½AY! " ;

} // BotonFinal

\_

Práctica 1.12: Instalación de Eclipse

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

18 Capítulo 1. Repaso de Java y Eclipse

Eclipse es un entorno integrado de desarrollo muy popular en el mundo

de Java. Está disponible en http://www.eclipse.org. La instalación varía

de un sistema a otro. Nosotros haremos una instalación manual sobre nuestro

sistema GNU/Linux, aunque es probable que la propia distribución (Debian,

Ubuntu, Max) ya tenga Eclipse paquetizado.

En la página hay disponibles varias versiones; debemos obtener la marcada

como \_Eclipse IDE for Java Developers\_, que contiene todo lo necesario

para desarrollar en Java. Una vez descargado, se obtiene un \_chero

eclipse-java-<version>.tar.gz. Descomprímelo en un directorio conocido

(la descompresión creará automáticamente el directorio eclipse donde

se meterán los archivos):

$ tar xfz $rutaDescarga/eclipse-java-<version>.tar.gz .

$ ./eclipse &

Cuando lo lanzamos, aparece la splash window durante la carga:

Luego nos pregunta la ruta donde queremos tener nuestro \_espacio de

trabajo\_. En el directorio que elijamos se guardarán los proyectos.

Al lanzarse eclipse, nos muestra el workspace vacío. Podemos abrir un

resumen de las características de Eclipse, las características nuevas de la

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

19

versión, ejemplos de proyectos, un tutorial de uso... Lo que queremos es comenzar

a trabajar, y para eso pulsamos sobre el icono de la derecha (Workbench).

Podremos volver a esta ventana en cualquier momento con Help - Welcome.

\_

Práctica 1.13: Hola mundo en Eclipse

Vamos a crear un primer proyecto en Java con eclipse:

Ve a File - New - Java Project

Establece como nombre del proyecto HolaMundo y pulsa Next.

Recorre las opciones prede\_nidas brevemente, y pulsa Finish.

Al volver, verás en el Package explorer el nuevo proyecto recién creado:

Tiene una carpeta src vacía.

Tiene una carpeta JRE System Library donde se enumeran las librer

ías disponibles en el JRE que se usará para lanzar la aplicación.

Vamos a crear el \_chero de código fuente.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

20 Capítulo 1. Repaso de Java y Eclipse

Pulsa sobre la carpeta src con el botón derecho, y selecciona New -

Class. También puedes hacerlo con el menú de la aplicación.

Como nombre de la nueva clase pon HolaMundo. Observa el resto de las

opciones y sus valores prede\_nidos. Verás aparecer un aviso de Eclipse

indicando que no es recomendable poner las clases en el paquete por

defecto (vacío). Como es una prueba, no nos preocuparemos.

Activa la casilla de veri\_cación para que Eclipse nos cree el método

main, y pulsa Finish.

Verás aparecer el esqueleto del código de la nueva clase. Escribe en el

main:

System.out.println("½Hola mundo!");

Para lanzar la aplicación, pulsa sobre el botón \_Play\_ de la barra de

herramientas, o en el menú Run - Run (Ctrl-F11). En la parte inferior

de Eclipse, donde se muestra la consola de ejecución, aparece el texto,

y el programa termina.

Mira en el directorio del proyecto y observa la aparición del \_chero

HolaMundo.class en el directorio bin. Eclipse se ha encargado de

compilar la clase por nosotros.

\_

Práctica 1.14: Depuración en Eclipse

Aunque para esta práctica no resulte de interés, vamos a ver brevemente

las capacidades de depuración de Eclipse. Modi\_ca el código ligeramente

para conseguir:

public clas s HolaMundo {

public s tat ic saluda ( St r ing nombre ) {

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

21

System . out . p r i n t l n ( " ½Hola " + nombre + " ! " ) ;

}

public s tat ic void main ( St r ing [ ] args ) {

saluda ( "As t e r ix " ) ;

saluda ( "Obel ix " ) ;

}

}

Vamos a utilizarlo para probar el depurador de Eclipse:

Haz doble click sobre el lateral izquierdo del código, para añadir un

punto de ruptura (breakpoint). También puedes colocar el cursor sobre

la línea y pulsar <Ctrl>-<Mays>-B. Verás aparecer un circulito

indicando la existencia del punto de ruptura.

Lanza la ejecución en depuración. Para eso puedes pulsar F11, o utilizar

el icono en la barra de herramientas:

Verás aparecer un aviso indicando que para depurar es preferible usar

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

22 Capítulo 1. Repaso de Java y Eclipse

la perspectiva de depuración, y nos pregunta si queremos pasar a ella.

Acepta la propuesta.

Date cuenta de que la ejecución se ha detenido en la línea donde pusiste

el punto de ruptura.

Pulsa F5, el icono Step into de la barra de herramientas, o la opción

del menú Run - Step Into.

Observa que el punto de parada ahora está en la primera línea del

método saluda(String). Además, en la ventana Debug donde está la

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

Notas bibliográ\_cas 23

\_pila de llamadas\_ verás los dos métodos (main y saluda) que están a

mitad de su ejecución.

Coloca el ratón sobre el identi\_cador nombre del código. Verás aparecer

una descripción de su contenido (Asterix).

Pulsa de nuevo F5 (Step into). Eclipse comenzará a ejecutar la invocaci

ón al System.out, entrando (y deteniéndose) en un código que no

es nuestro. Para ejecutar hasta terminar el método actual, pulsa F7

(Step Return), o usa el botón de la barra de herramientas, cercano al

de Step Into.

Vuelve a pulsar F7 para terminar la ejecución de la primera invocación

a saluda. Comprueba en la ventana Console la aparición del texto

½Hola Asterix!.

El programa habrá vuelto al main. Date cuenta de que en la ventana

Debug ha desaparecido el método.

Pulsa F6 (Step Over) o el correspondiente icono en la barra de herramientas

para ejecutar la línea actual completamente, sin detenerse

en la ejecución paso a paso de las invocaciones. No entraremos en el

método saluda, que se ejecutará completamente.

Habrá aparecido el segundo saludo. Pulsa F8 (Resume) para ejecutar

sin volverse a detener1.

\_

Notas bibliográ\_cas

La cantidad de documentación disponible sobre Java es inmensa. Para

comenzar, quizá un buen punto de partida sean los tutoriales proporcionados

por Oracle (http://docs.oracle.com/javase/tutorial/index.html) o, si

se pre\_eren los libros tradicionales, la última edición del conocido Piensa en

Java de Bruce Eckel.

Los applets vivieron su explendor a \_nales de la década de 1990 y principios

de la del 2000. Hoy su uso es minoritario, habiendo sido desplazados

por los modos de interacción con el navegador que han proporcionado las

1Este comando apenas es útil en este ejemplo porque estábamos terminando, y no

habría diferencia con usar cualquiera de los demás comandos. Sin embargo en situaciones

más complejas es muy práctico para relanzar la ejecución hasta el siguiente punto de

ruptura.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín

24 Capítulo 1. Repaso de Java y Eclipse

págianas dinámicas y JavaScript. En este capítulo nos ha interesado principalmente

su ciclo de vida, y el hecho de que se ejecutan integrados en un

navegador. Se puede encontrar más información sobre ellos por ejemplo en

http://docs.oracle.com/javase/tutorial/deployment/applet/.

También en los tutoriales de Oracle hay una sección dedicada especialmente

a las clases anidadas (http://docs.oracle.com/javase/tutorial/

java/javaOO/nested.html).

Eclipse es un IDE ampliamente utilizado por los desarrolladores de Java.

Sin embargo, en ocasiones puede ser bastante complejo, debido a su tama-

ño y versatilidad. Afortunadamente, incluye una ayuda relativamente rica,

que incluye incluso tutoriales para iniciarse en su uso. En el sitio web del

programa (http://www.eclipse.org) también se pueden encontrar algunos

tutoriales.

En el próximo capítulo. . .

En el próximo capítulo daremos nuestros primeros pasos con Android,

instalando el software necesario para desarrollar aplicaciones para él, y analizando

sus herramientas principales.

IFC03CM12 - Programación de dispositivos móviles - Julio 2012

Pedro Pablo Gómez Martín