

Programación orientada a objetos

Capítulo 5

Comportamiento más sofisticado

Tema 5. Comportamiento avanzado con objetos. Semana 5

- 1- Documentación de las clases de una librería
- 2- Los paquetes y la sentencia import
- 3- Visibilidad
 - a. Ocultamiento de la información
 - b. Métodos y campos públicos y privados
- 4- Variables de clase y constantes
 - a. La palabra clave static
 - b. Constantes

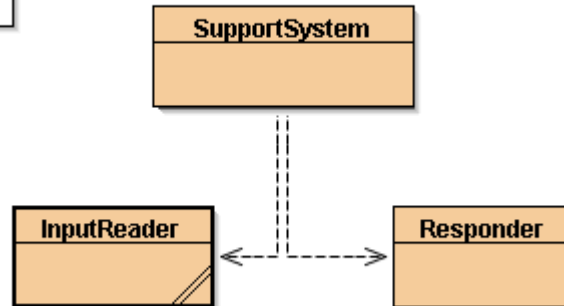
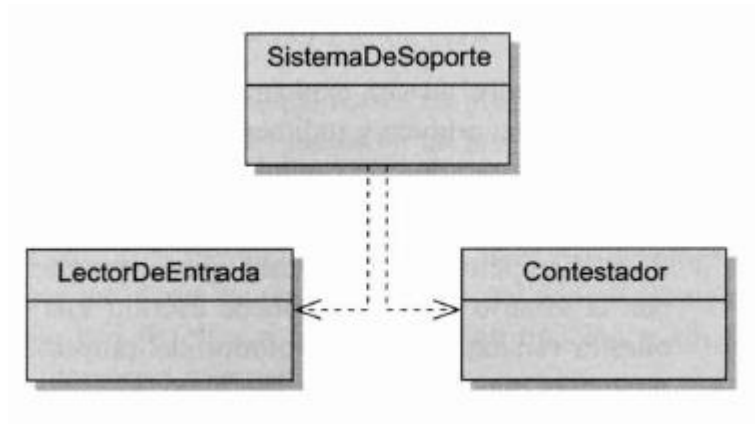
- 1- Estudiar el capítulo 5 del libro base para la Unidad Didáctica I
- 2- Leer el Apéndice I del libro base para la Unidad Didáctica I
- 4- Realizar los ejercicios en el entorno BlueJ sugeridos en el libro base
- 3- Incluir la documentación en el código de la práctica y encapsular los campos

5.1 Documentación para clases de librería

Biblioteca Java. La biblioteca de clases estándar de Java contiene muchas clases que son muy útiles. Es importante saber cómo se usa la biblioteca.

- Para hacer un programa orientado a objetos
 - 1. Identificar los objetos del dominio
 - 2. Ver cómo se pueden implementar usando clases ya existentes
 - Evitar reinventar la rueda
 - Las clases estándar suelen estar muy probadas y son eficientes
 - Para ello es conveniente familiarizarse con las librerías estándar de Java (o del lenguaje correspondiente)
 - La librería está formada por miles de clases con sus métodos, parámetros y tipos de retornos (si tiene)
 - Es necesario conocer bien clases de uso común (por su nombre)
 - Saber encontrar información acerca de las demás clases
 - Lo más importante es la interfaz, no la implementación
 - Cómo usar una clase, no cómo se implementa

5.2 El sistema “Soporte técnico”



```
BlueJ: Terminal Window - soporte-tecnico1
Options
Bienvenido al Sistema de Soporte Técnico de DodgySoft.

Por favor, cuéntenos su problema.
Lo asistiremos con cualquier problema que tenga.
Para salir del sistema escriba 'bye'.
> Despues de iniciarlo, mi sistema siempre se cae
Lo que dice parece interesante, cuénteme un poco más...
> Tengo Windows 3000. Su programa, ¿corre en Windows 3000?
Lo que dice parece interesante, cuénteme un poco más...
> No puedo usar su software. Necesito ayuda!
Lo que dice parece interesante, cuénteme un poco más...
> ¿Por que siempre dice "Lo que dice parece interesante"?
Lo que dice parece interesante, cuénteme un poco más...
> bye
Un gusto hablar con Ud. Bye...
```

Clase “SistemaDeSoporte”

```
public class SistemaDeSoporte
{
    private LectorDeEntrada lector;

    private Contestador contestador;

    /**
     * Crea un sistema de soporte técnico.
     */
    public SistemaDeSoporte()
    {
        lector = new LectorDeEntrada();
        contestador = new Contestador();
    }
}
```

```
public void iniciar()
{
    boolean terminado = false;
    imprimirBienvenida();
    while(!terminado) {
        String entrada = lector.getEntrada();
        if(entrada.startsWith("bye")) {
            terminado = true;
        }
        else {
            String respuesta =
contestador.generarRespuesta();
            System.out.println(respuesta);
        }
    }
    imprimirDespedida();
}
```

Clase "SistemaDeSoporte"

```
    * Imprime un mensaje de bienvenida en la pantalla.
    */
    private void imprimirBienvenida()
    {
        System.out.println(
            "Bienvenido al Sistema de Soporte Técnico
de DodgySoft.");
        System.out.println();
        System.out.println("Por favor, cuéntenos su
problema.");
        System.out.println(
            "Lo asistiremos con cualquier problema que
tenga.");
    }
```

```
        System.out.println("Para salir del sistema
escriba 'bye'.");
    }
    /**
    * Imprime un mensaje de despedida en la pantalla.
    */
    private void imprimirDespedida()
    {
        System.out.println("Un gusto hablar con Ud.
Bye...");
    }
}
```

Clase “Contestador”

```
/**
 * La clase contestador representa un objeto generador de
 respuestas.
 * Se lo usa para generar una respuesta automatizada.
 *
 * @author      Michael Kölling y David J. Barnes
 * @version     0.1   (2006.03.30)
 */
public class Contestador
{
    /**
     * Construye un Contestador, no hay nada para hacer.
     */
    public Contestador()
    {
    }
    /**
     * Genera una respuesta.
     * @return     Una cadena que se mostrará como una
 respuesta
     */
    public String generarRespuesta()
    {
        return "Lo que dice parece interesante,
 cuénteme un poco más...";
    }
}
```


5.3 Lectura de documentación de clase

- <http://docs.oracle.com/javase/7/docs/api/>

The screenshot shows a Mozilla Firefox browser window displaying the Java 2 Platform Standard Edition 5.0 API Specification. The browser's address bar shows the URL docs.oracle.com/javase/1.5.0/docs/api/index.html. The page title is "Overview (Java 2 Platform SE 5.0) - Mozilla Firefox". The browser's menu bar includes "Archivo", "Editar", "Ver", "Historial", "Marcadores", "Herramientas", and "Ayuda". The browser's toolbar includes "Más visitados", "Comenzar a usar Firefox", "Últimas noticias", "Compartir en Facebook", "Barra de contactos", and "save on delicious".

The page content includes a navigation bar with "Overview", "Package", "Class", "Use", "Tree", "Deprecated", "Index", and "Help". The main heading is "Java™ 2 Platform Standard Edition 5.0 API Specification". Below the heading, it states: "This document is the API specification for the Java 2 Platform Standard Edition 5.0." and "See: [Description](#)".

The "All Classes" sidebar lists the following classes: [AbstractAction](#), [AbstractBorder](#), [AbstractButton](#), [AbstractCellEditor](#), [AbstractCollection](#), [AbstractColorChooserPane](#), [AbstractDocument](#), [AbstractDocument.Attribute](#), [AbstractDocument.Content](#), [AbstractDocument.Element](#), [AbstractExecutorService](#), [AbstractInterruptibleChannel](#), [AbstractLayoutCache](#), [AbstractLayoutCache.Node](#), [AbstractList](#), [AbstractListModel](#), [AbstractMap](#), and [AbstractMethodError](#).

The "Java 2 Platform Packages" table lists the following packages and their descriptions:

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
	Drag and Drop is a direct manipulation gesture found in many

The browser's status bar at the bottom shows "zotero", "RefGrab-It", and "WR".

5.3.1 Comparar interfaz e implementación

La **interfaz** de una clase describe lo que es capaz de hacer dicha clase y la manera en que se puede usar sin mostrar su implementación.

El código completo que define una clase se denomina la **implementación** de dicha clase.

Objetos inmutables.

Se dice que un objeto es inmutable si su contenido o su estado no puede ser cambiado una vez que se ha creado. Los objetos **String** son un ejemplo de objetos inmutables.

Habrás visto que la documentación incluye diferentes piezas de información, entre otras:

- el nombre de la clase
- una descripción general del propósito de la clase;
- una lista de los constructores y los métodos de la clase;
- los parámetros y los tipos de retorno de cada constructor y de cada método;
- una descripción del propósito de cada constructor y cada método.

También se utiliza la terminología interfaz referida a métodos individuales. Por ejemplo, la documentación de la clase `String` nos muestra la interfaz del método `length`:

```
public int length2()  
    Returns the length of this string. The length is equal  
    to the number of 16-bit Unicode characters in the  
    string.
```

```
Returns:  
the length of the sequence of characters represented by  
this object.
```

La interfaz de un método consiste en su *signatura* y un comentario (que se muestra en el ejemplo en letra cursiva). La signatura de un método incluye, en este orden:

- un modificador de acceso que discutiremos más adelante (en este caso, `public`);
- el tipo de retorno del método (en este caso, `int`);
- el nombre del método;
- una lista de parámetros (que en este caso es vacía).

La interfaz de un método proporciona todos los elementos necesarios para saber cómo usarlo.

5.3.2 Usar métodos de clases de librería

```
entrada = entrada.trim();
```

Este código le solicita al objeto almacenado en la variable `entrada` crear una nueva cadena similar a la dada, pero eliminados los espacios en blanco antes y después de la palabra. Luego la nueva cadena se almacena en la variable `entrada` por lo que pierde su viejo contenido, y en consecuencia, después de esta línea de código, `entrada` hace referencia a una cadena sin espacios al inicio y al final.

Ahora podemos insertar esta línea en nuestro código de modo que quede así:

```
String entrada = lector.getEntrada();
entrada = entrada.trim();
if (entrada.startsWith("bye")) {
    terminado = true;
}
else {
    Se omitió el código
}
```

Las primeras dos líneas podrían unirse para formar una sola línea:

```
String entrada = lector.getEntrada().trim();
```

```
(lector.getEntrada()).trim()
```

5.3.3 Comprobar la igualdad de cadenas

Cuidado: la comparación de dos cadenas mediante el operador `==` puede producir resultados incomprensibles e inesperados. Como regla general, las cadenas siempre se pueden comparar mediante el método `equals` en lugar de hacerlo con el operador `==`.

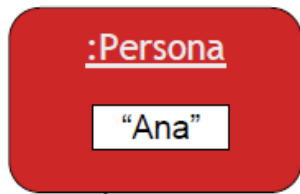
```
if (entrada == "bye") {    // ¡no siempre funciona!
    ...
}
```

El problema aquí radica en que es posible que existan varios objetos `String` independientes que representen la misma cadena. Por ejemplo, dos objetos `String` podrían contener ambos los caracteres «bye». ¡El operador (`==`) evalúa si ambos operandos hacen referencia al mismo objeto, no si sus valores son iguales! Y esta es una diferencia importante.

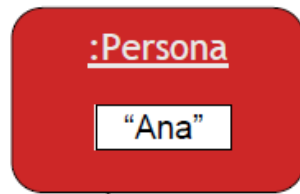
La solución para este problema es usar el método `equals` definido en la clase `String`. Este método comprueba correctamente si dos objetos `String` tienen el mismo contenido. El código correcto es el siguiente:

```
if (entrada.equals("chau")) {
    ...
}

System.out.println (n1 == n2);           //false
System.out.println (n1.equals(n2));     //true
```

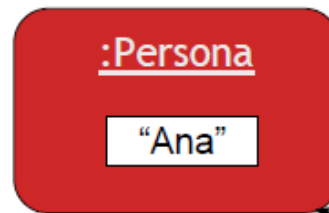


personal

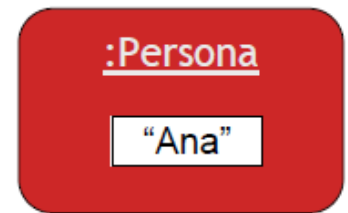


persona2

personal == persona2 ?



personal



persona2

personal == persona2 ?

Igualdad vs Identidad

```
class PruebaIgualdad {
    public static void main(String args[]) {
        String str1, str2;
        str1 = "Texto de prueba.";
        str2 = str1;

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("El mismo objeto? " + (str1 == str2));

        str2 = new String(str1);

        System.out.println("String1: " + str1);
        System.out.println("String2: " + str2);
        System.out.println("El mismo objeto? " + (str1 == str2));
        System.out.println("El mismo valor? " + str1.equals(str2));
    }
}
```

5.4 Agregar comportamiento aleatorio

Aleatorio y pseudo-aleatorio: la generación de números por azar mediante una computadora no es en realidad tan fácil como uno podría pensar. Las computadoras operan de una manera bien definida y determinística que se apoya en el hecho de que todo cálculo es predecible y repetible, en consecuencia, existe poco espacio para un comportamiento realmente aleatorio.

Los investigadores, a lo largo del tiempo, han propuesto muchos algoritmos para producir secuencias semejantes a los números aleatorios. Estos números no son típicamente números aleatorios verdaderos, pero siguen reglas muy complicadas. Estos números se conocen como números *pseudo-aleatorios*.

En un lenguaje como Java, afortunadamente, la generación de números pseudo-aleatorios ha sido implementada en una clase de la biblioteca, de modo que, todo lo que tenemos que hacer para obtener un número de este tipo es escribir algunas invocaciones a dicha biblioteca.

5.4.1 La clase Random

Para generar un número aleatorio tenemos que:

- crear una instancia de la clase Random y
- hacer una llamada a un método de esa instancia para obtener un número.

```
Random generadorDeAzar;  
  
generadorDeAzar = new Random();  
int indice = generadorDeAzar.nextInt();  
System.out.println(indice);
```

La clase Random también ofrece un método que soporta esta restricción, su nombre también es `nextInt`, pero tiene un parámetro para especificar el rango de números que queremos usar.

Cuando se utiliza un método para generar números por azar en un rango especificado, debe tenerse el cuidado de verificar si los límites se incluyen o no en el del intervalo. El método `nextInt(int n)` de la clase Random de la biblioteca de Java especifica que genera números desde 0 (inclusive) hasta n (exclusive). Esto quiere decir que el valor 0 está incluido entre los posibles valores de los resultados, mientras que el valor especificado por n no está incluido. El máximo número posible que devuelve es n-1.

5.4.3 Clase Contestador

```
public Contestador()
{
    generadorDeAzar = new Random();
    respuestas = new ArrayList<String>();
    rellenarRespuestas();
}
```

```
public String generarRespuesta()
{
    // Toma un número aleatorio para el índice de
la lista
    // de respuestas por defecto.
    // El número estará entre 0(inclusive) y el
tamaño de
    // la lista(exclusive).
    int indice =
generadorDeAzar.nextInt(respuestas.size());
    return respuestas.get(indice);
}
```

```

private void rellenarRespuestas()
{
    respuestas.add("Parece complicado. ¿Podría
describir \n" +
                    "el problema más
detalladamente?");
    respuestas.add("Hasta ahora, ningún cliente
informó \n" +
                    "sobre este problema.
\n" +
                    "¿Cuál es la
configuración de su equipo?");
    respuestas.add("Lo que dice parece interesante,
\n" +
                    "cuénteme un poco más...
");
    respuestas.add("Necesito un poco más de
información. \n");
    respuestas.add("¿Verificó si tiene algún
conflicto \n" +
                    "con una dll? \n" );
    respuestas.add("Ese problema está explicado en
el manual. \n" +
                    "¿Leyó el manual? ");
    respuestas.add("Su descripción es un poco
confusa. \n" +
                    "¿Cuenta con algún
experto que lo \n" +
                    "ayude a describir el
problema \n" +
                    "de manera más
precisa?");
    respuestas.add("Eso no es una falla, es una
característica \n" +
                    "del programa. \n" );
    respuestas.add("¿Ha podido elaborar esto?");
}
}

```

5.5 Paquetes y la sentencia “import”

Las clases de Java se almacenan en la biblioteca de clases pero no están disponibles automáticamente para su uso, tal como las otras clases del proyecto actual. Para poder disponer de alguna de estas clases, debemos explicitar en nuestro código que queremos usar una clase de la biblioteca. Esta acción se denomina *importación de la clase* y se implementa mediante la sentencia `import`. La sentencia `import` tiene la forma general

```
import nombre-de-clase-calificado;  
  
import java.util.ArrayList;  
import java.util.Random;
```

Java también nos permite importar paquetes completos con sentencias de la forma

```
import nombre-del-paquete.*;
```

Por lo que la siguiente sentencia importaría todas las clases del paquete `java.util`:

```
import java.util.*;
```

5.6.1 Concepto de Mapa

Un **mapa** es una colección que almacena pares llave/valor como entradas. Los valores se pueden buscar suministrando la llave.

Un mapa es una colección de pares de objetos llave/valor. Tal como el `ArrayList`, un mapa puede almacenar un número flexible de entradas. Una diferencia entre el `ArrayList` y un `Map` es que, en un `Map` cada entrada no es un único objeto sino un *par* de objetos. Este par está compuesto por un objeto *llave* y un objeto *valor*.

En lugar de buscar las entradas en esta colección mediante un índice entero (como hicimos con el `ArrayList`) usamos el objeto *llave* para buscar el objeto *valor*.

5.6.2 HashMap

Un HashMap es una implementación particular de un Map. Los métodos más importantes de la clase HashMap son put y get.

```
HashMap<String, String> agenda = new HashMap<String, String>();  
agenda.put("Charles Nguyen", " (531) 9392 4587");  
agenda.put("Lisa Jones", " (402) 4536 4674");  
agenda.put("William H. Smith", " (998) 5488 0123");
```

El siguiente código busca el número de teléfono de Lisa Jones y lo imprime:

```
String numero = agenda.get("Lisa Jones");  
System.out.println(numero);
```

¿Qué ocurre si buscamos(get) una clave que no está?
¿Qué ocurre si introducimos de nuevo (put) una clave existente?

Get: Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

Put: If the map previously contained a mapping for the key, the old value is replaced.

5.7 Usar conjuntos

Un **conjunto** es una colección que almacena cada elemento individual una sola vez como máximo. No mantiene un orden específico.

```
import java.util.HashSet;
import java.util.Iterator;
...
HashSet<String> miConjunto = new HashSet<String>();

miConjunto.add("uno");
miConjunto.add("dos");
miConjunto.add("tres");
```

List, Map y Set Es tentador asumir que se puede usar un **HashSet** de manera similar a un **HashMap**. En realidad, tal como lo ilustramos, la forma de usar un **HashSet** es más parecida a la forma de usar un **ArrayList**. Cuando tratamos de comprender la forma en que se usan las diferentes clases de colecciones, la segunda parte del nombre es la mejor indicación de los datos que almacenan, y la primera palabra describe la forma en que se almacenan. Generalmente estamos más interesados en el «qué» (la segunda parte) antes que en el «cómo». De modo que un **TreeSet** debiera usarse de manera similar a un **HashSet**, mientras que un **TreeMap** debiera usarse de manera similar a un **HashMap**.

5.8 Dividir Cadenas

```
/**
 * Lee una línea de texto desde la entrada estándar
 (la terminal de
 * texto) y la retorna como un conjunto de palabras.
 *
 * @return Un conjunto de cadenas en el que cada
String es una de las
 * palabras que escribió el usuario.
 */
public HashSet<String> getEntrada()
{
    System.out.print("> "); //
imprime el prompt
    String linea =
lector.lineaSiguiete().trim().toLowerCase();

    String[] arregloDePalabras = linea.split(" ");

    // agrega las palabras del arreglo en el
hashset
    HashSet<String> palabras = new HashSet<String>();
    for (String palabra : arregloDePalabras) {
        palabras.add(palabra);
    }

    return palabras;
}
```

El método `split` puede dividir una cadena en distintas subcadenas y las devuelve en un arreglo de cadenas. El parámetro del método `split` establece la clase de caracteres de la cadena original que producirá la división en palabras. Hemos determinado que queremos dividir nuestra cadena mediante cada carácter espacio en blanco.

Sistema TechSupport

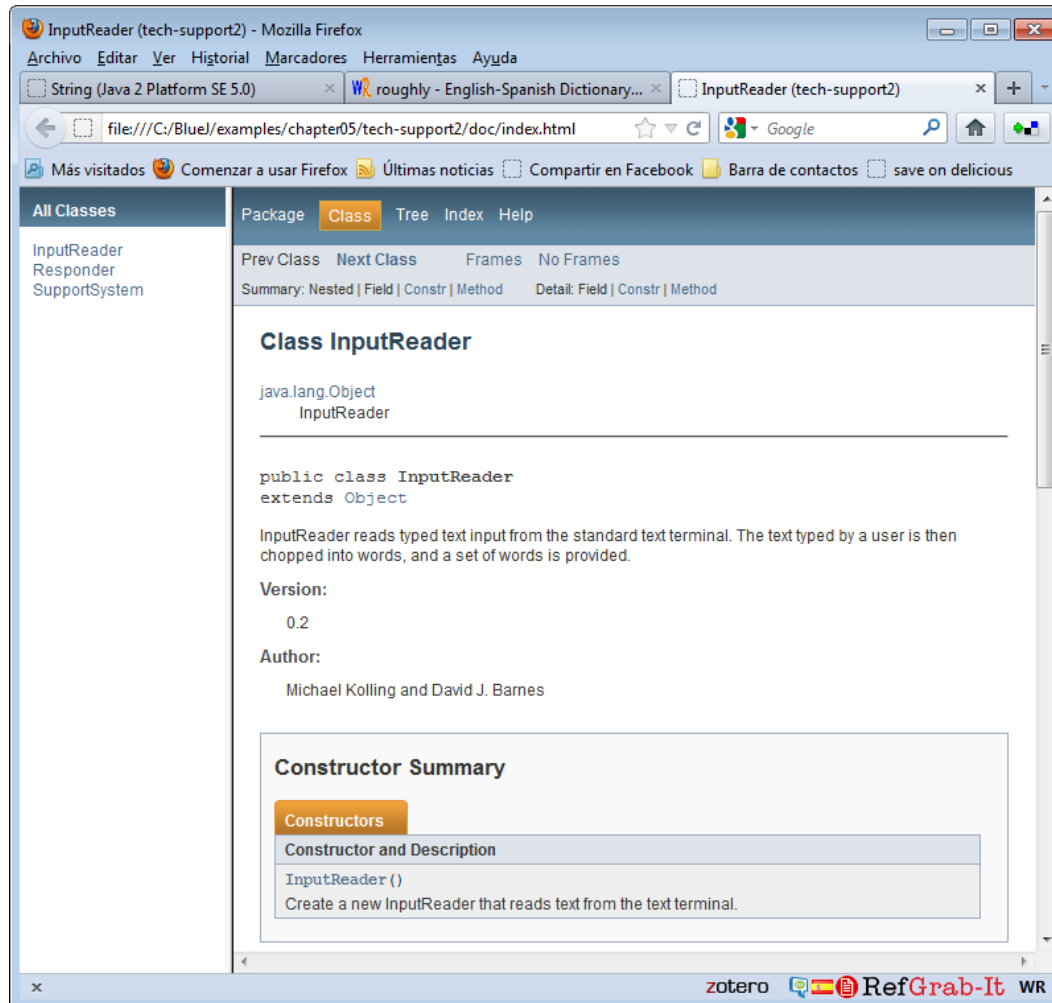
```
public void iniciar()
{
    boolean terminado = false;
    imprimirBienvenida();
    while(!terminado) {
        HashSet<String> entrada =
(lector.getEntrada());
        if(entrada.contains("bye")) {
            terminado = true;
```

```
        }
        else {
            String respuesta =
contestador.generarRespuesta(entrada);
            System.out.println(respuesta);
        }
    }
    imprimirDespedida();
}
```

```
public String generarRespuesta(HashSet<String> palabras)
{
    Iterator<String> it = palabras.iterator();
    while (it.hasNext()) {
        String palabra = (String) it.next();
        String respuesta =
mapaDeRespuestas.get(palabra);
        if (respuesta != null) {
            return respuesta;
        }
        // si llega acá es porque la palabra no fue
reconocida
        // En este caso, tomamos una de nuestras
respuestas por defecto
        return getRespuestaPorDefecto();
    }
}
```


5.10 Escribir documentación de clase

- (BlueJ) Herramientas -> documentación proyecto



The screenshot shows a Mozilla Firefox browser window displaying the JavaDoc documentation for the `InputReader` class. The browser's address bar shows the file path `file:///C:/BlueJ/examples/chapter05/tech-support2/doc/index.html`. The page content includes:

- Class InputReader**
- Package: `java.lang.Object`
- Class: `InputReader`
- Code snippet:

```
public class InputReader extends Object
```
- Description: `InputReader` reads typed text input from the standard text terminal. The text typed by a user is then chopped into words, and a set of words is provided.
- Version: 0.2
- Author: Michael Kolling and David J. Barnes
- Constructor Summary**
- Constructors section with a table:

Constructor and Description
<code>InputReader ()</code> Create a new <code>InputReader</code> that reads text from the text terminal.

5.10.2 Elementos de la documentación de una clase

La documentación de una clase debe incluir como mínimo:

- el nombre de la clase;
- un comentario que describa el propósito general y las características de la clase;
- un número de versión;
- el nombre del autor (o de los autores);
- la documentación de cada constructor y de cada método.

La documentación de cada constructor y de cada método debe incluir:

- el nombre del método;
- el tipo de retorno;
- los nombres y tipos de los parámetros;
- una descripción del propósito y de la función del método;
- una descripción de cada parámetro;
- una descripción del valor que devuelve.

5.11 Compara público con privado

Los modificadores de acceso son las palabras clave `public` o `private` que aparecen al comienzo de las declaraciones de campos y de las firmas de los métodos. Por ejemplo:

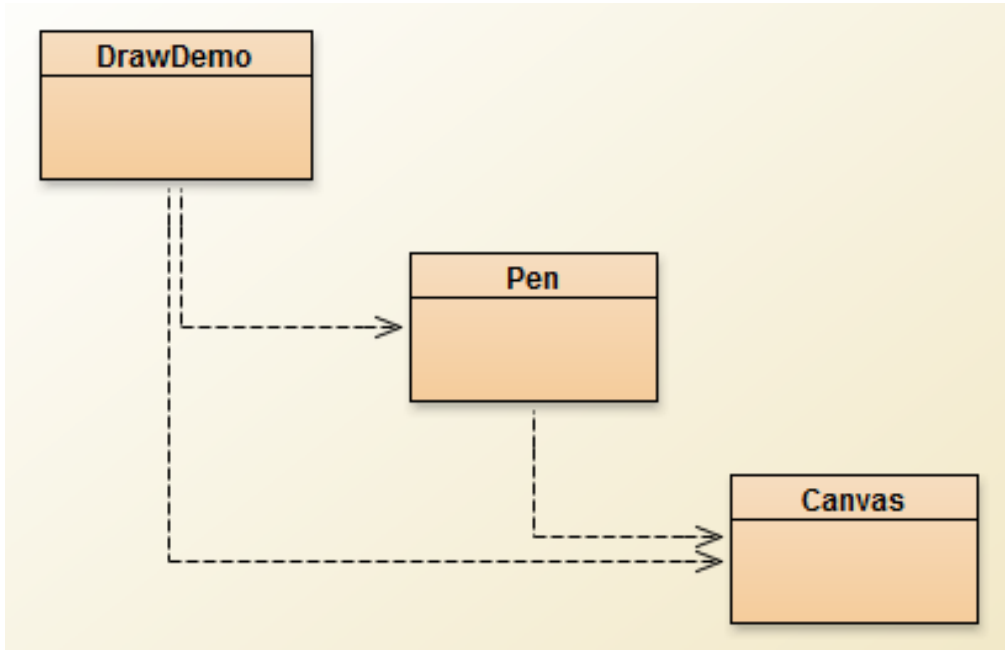
```
// declaración de campo
private int numeroDeAsientos;
// métodos
public void setEdad(int nuevaEdad)
{ ...
}
private int calcularPromedio()
{...
}
```

El **ocultamiento de la información** es un principio que establece que los detalles internos de implementación de una clase deben permanecer ocultos para las otras clases. Asegura una mejor modularización de la aplicación.

Los **modificadores de acceso** definen la visibilidad de un campo, de un constructor o de un método. Los elementos públicos son accesibles dentro de la misma clase o fuera de ella; los elementos privados son accesibles solamente dentro de la misma clase.

Modificador	Se aplica a	Ella misma	Package	Hijas	Otras clases
public	clases, interfaces, métodos y variables	✓	✓	✓	✓
protected	métodos y variables	✓	✓	✓	
	clases, interfaces, métodos y variables	✓	✓		
private	métodos y variables	✓			
private protected *	métodos y variables	✓		✓	

5.12 Proyecto scribble



Finalización de código

```
import java.awt.Color;
import java.util.Random;

/**
 * Class DrawDemo - provides some short demonstrations showing how to use the
 * Pen class to create various drawings.
 *
 * @author Michael Kölling and David J. Barnes
 * @version 2011.07.31
 */

public class DrawDemo
{
    private Canvas myCanvas;
    private Random random;

    mycanvas.
    /** void clear()
     * Pre Object clone()
     */
    public void colorScribble()
    {
        my void drawSquare()
        ra void drawWheel()
    }
    boolean equals(Object)
    void finalize()
    Class<?> getClass()
    /** int hashCode()
     * Dra void notify()
     */
    public void notifyAll()
    {
        Pen pen = new Pen(320, 260, myCanvas);
        pen.setColor(Color.BLUE);
    }
}
```

DrawDemo

void clear()

Clear the screen.

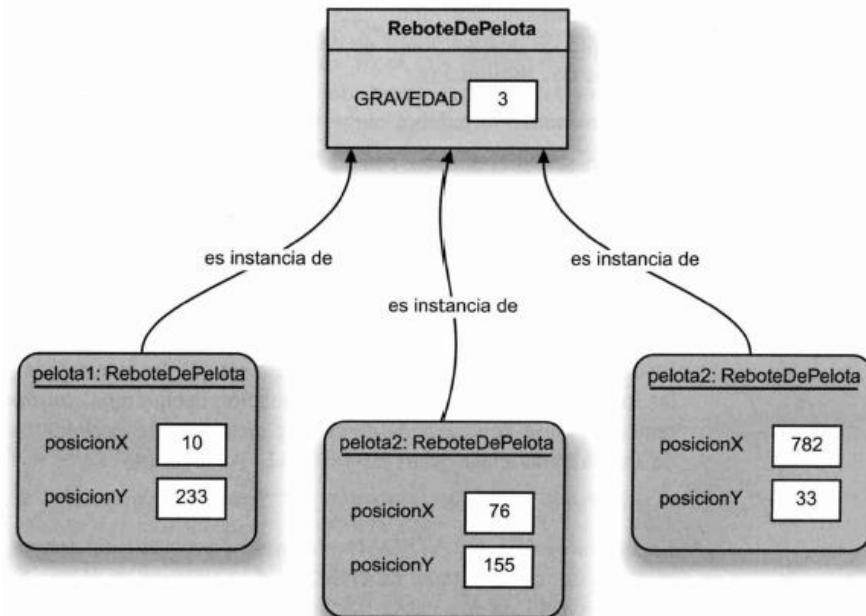
Ctrl + espacio

5.13 Variables de clases: la palabra clave “static”

Las clases pueden tener campos: estos campos se conocen como **variables de clase** o variables estáticas. En todo momento, existe exactamente una copia de una variable de clase, independientemente del número de instancias que se hayan creado.

```
public class ReboteDePelota
{
    // Efecto de gravedad
    private static final int GRAVEDAD = 3;

    private int posicionX;
    private int posicionY;
    Se omiten otros campos y métodos
}
```



5.13.2 Constantes

- deben incluir la palabra clave `final` antes del nombre del tipo y
- deben ser inicializadas con un valor en el momento de su declaración.

```
private final int TOPE = 10;
```

Constantes de “clase”

En la práctica, es muy frecuente el caso en que las constantes se relacionen con todas las instancias de una clase. En esta situación declaramos *constantas de clase*. Las constantes de clase son campos de clase constantes. Se declaran usando una combinación de las palabras clave `static` y `final`. Por ejemplo:

```
private static final int TOPE = 10;
```


Miembro estático

```
public class Cuenta{  
    public static String INFO = "Ejemplo de clase Cuenta";  
    public final static int UNO = 1;  
    public final static int DOS = 2;  
    public final static int TRES = 3;  
}
```

- Se pueden acceder
 - Directamente desde la clase

```
System.out.println(Cuenta.UNO);
```



Consola

```
1
```

- O bien desde un objeto cualquiera

```
Cuenta cuenta = new Cuenta();  
System.out.println(cuenta.INFO);
```



Consola

```
Ejemplo de clase Cuenta
```

Términos introducidos en este capítulo

intertaz, implementación, mapa, conjunto, javadoc, modificador de acceso, ocultamiento de información, acoplamiento, variable de clase, estático, constante, final

Resumen de conceptos

- **biblioteca de Java** La biblioteca de clases estándar de Java contiene muchas clases que son muy útiles. Es importante saber cómo usar la biblioteca.
- **documentación de la biblioteca** La documentación de la biblioteca estándar de Java muestra detalles sobre todas las clases de la biblioteca. El uso de esta documentación es esencial para hacer un buen uso de las clases de la biblioteca.
- **interfaz** La interfaz de una clase describe lo que hace la clase y cómo puede usarse sin mostrar su implementación.
- **implementación** El código completo que define una clase se denomina implementación de dicha clase.
- **inmutable** Se dice que un objeto es inmutable si su contenido o su estado no puede ser modificado una vez que fue creado. Los Strings son ejemplos de objetos inmutables.
- **mapa** Un mapa es una colección que almacena entradas de pares de valores llave/valor. Los valores pueden ser buscados mediante el suministro de una llave.
- **conjunto** Un conjunto es una colección que almacena cada elemento una única vez. No mantiene ningún orden específico.

- **documentación** La documentación de una clase debe ser suficientemente detallada como para que otros programadores puedan usar la clase sin necesidad de leer su implementación.
- **modificadores de acceso** Los modificadores de acceso definen la visibilidad de un campo, un constructor o un método. Los elementos públicos son accesibles dentro de la misma clase y desde otras clases; los elementos privados son accesibles solamente dentro de la misma clase a la que pertenecen.
- **ocultamiento de la información** El ocultamiento de la información es un principio que establece que los detalles internos de la implementación de una clase deben permanecer ocultos para las otras clases. Asegura la mejor modularización de una aplicación.
- **variables de clase, variables estáticas** Las clases pueden tener campos que se conocen como variables de clase o variables estáticas. En todo momento, existe una única copia de una variable de clase, independientemente del número de instancias que se hayan creado.