Programación orientada a objetos

Capítulo 6

Comportamiento más sofisticado

Tutor: Manuel Fernández Barcell
Centro Asociado de Cádiz
http://prof.mfbarcell.es

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

Tema 5. Se ocupa de las bibliotecas de clases y las interfaces. En concreto se presenta la biblioteca estándar de Java, donde se discuten algunas de sus clases más relevantes. Así mismo, este tema pretende mostrar al alumno el modo en el que toda esta documentación de referencia puede ser accedida y consultada.

Tema 5:

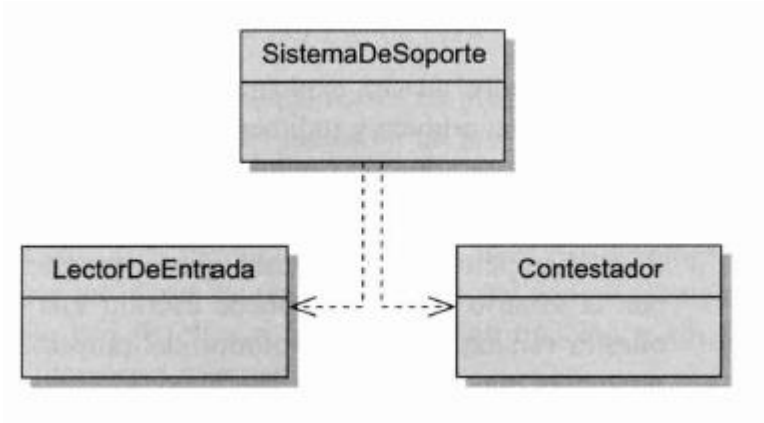
1- Documentar una clase correctamente utilizando JavaDoc como estándar.
2- Ser capaz de distinguir entre campos que deban ser privados o públicos en función del problema, así como detectar aquellas situaciones en las que una clase deba definirse como estática.

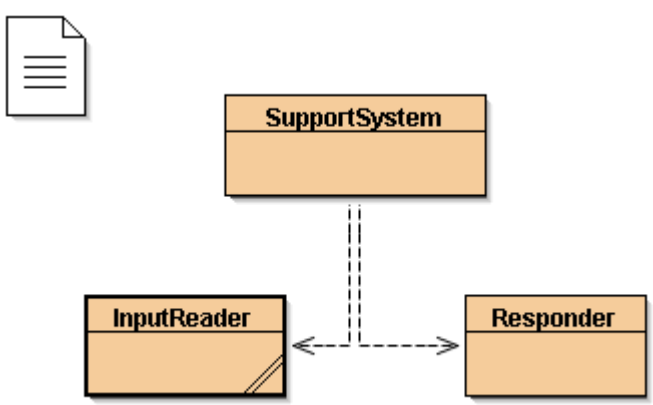
6.1 Documentación para clases de librería

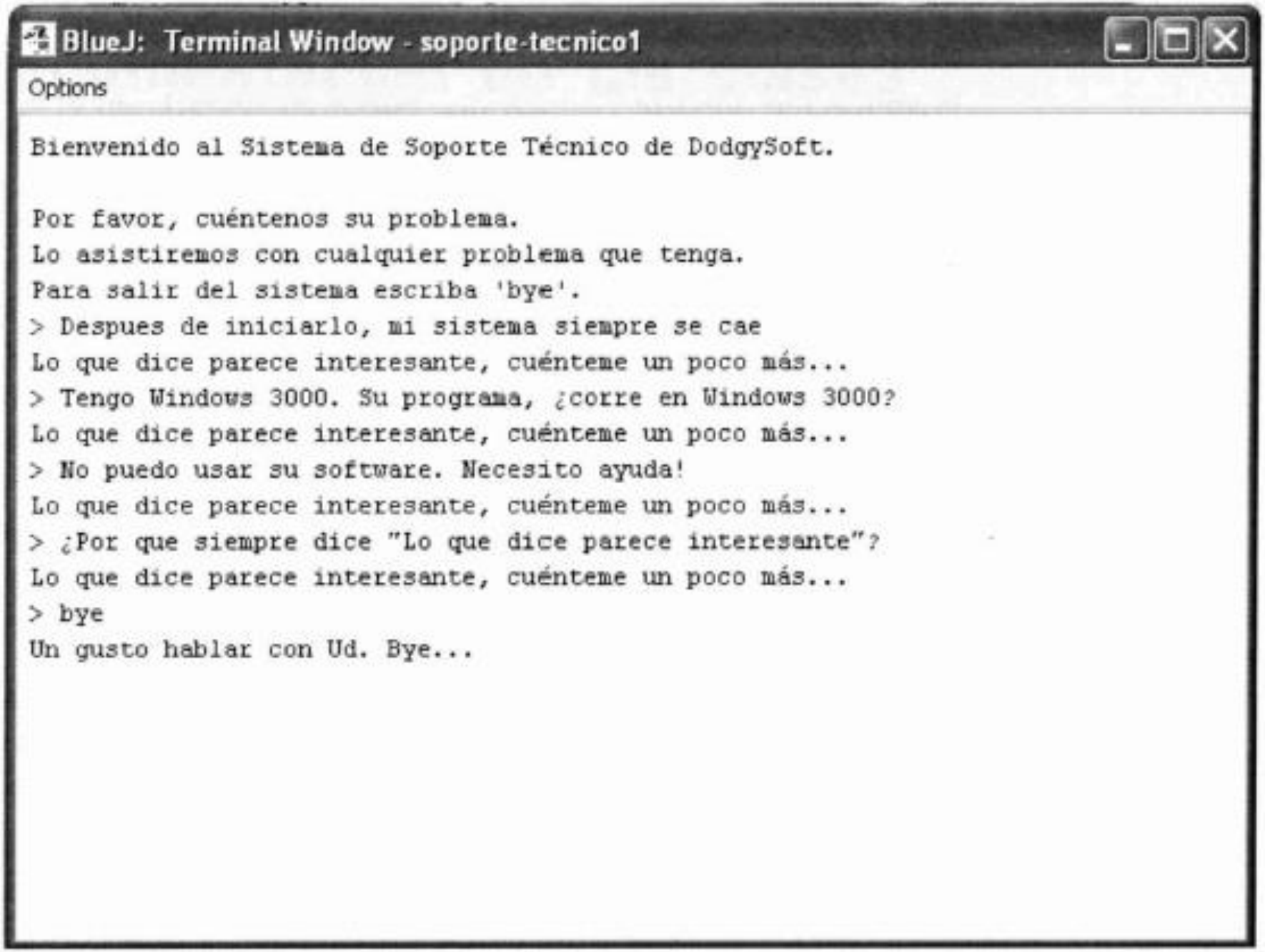
Biblioteca Java. La biblioteca de clases estándar de Java contiene muchas clases que son muy útiles. Es importante saber cómo se usa la biblioteca.

- 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)
    - Un buen programador de Java
      - Debe conocer bien clases de uso común (por su nombre)
      - Debe saber encontrar información acerca de las demás clases
  - Lo más importante es la interfaz de la clase, no la implementación concreta
    - Cómo usar una clase (el nombre de la clase, los nombres de los métodos, los parámetros, los tipos de retornos) no cómo se implementa
- Los objetos son instancias de las clases
  - Cada objeto tiene sus propios valores de las variables definidas en la clase
  - Cada objeto tiene su propia identidad

6.2 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.");
```

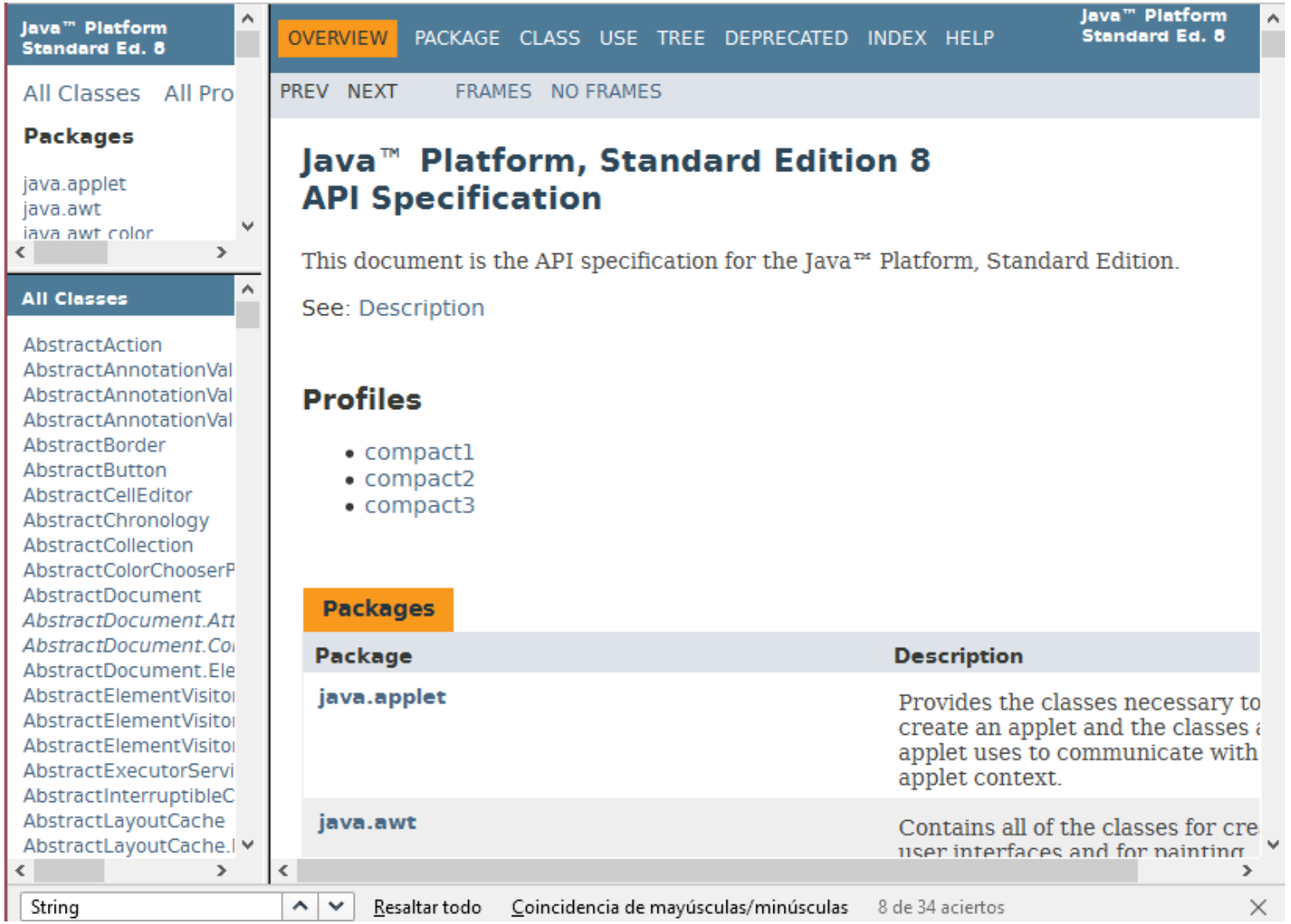
```
            System.out.println("Para salir del sistema
 escriba 'bye'.");
      }
      /**
       * Imprime un mensaje de despedida en la pantalla.
       */
      private void imprimirDespedida()
      {
            System.out.println("Un gusto hablar con Ud.
 Bye...");
      }
 }
```

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...";
    }
}
```

6.3 Lectura de documentación de clase

- http://docs.oracle.com/javase/7/docs/api/
- https://docs.oracle.com/javase/8/



6.3.1 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.

Habrá 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.

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.

Pitfall It is a common error in Java to try to modify a string—for example by writing

```
input.toUpperCase();
```

This is incorrect (strings cannot be modified), but this unfortunately does not produce an error. The statement simply has no effect, and the input string remains unchanged.

The `toUpperCase` method, as well as other string methods, does not modify the original string, but instead returns a *new* string that is similar to the original one, with some changes applied (here, with characters changed to uppercase). If we want our input variable to be changed, then we have to assign this new object back into the variable (discarding the original one), like this:

```
input = input.toUpperCase();
```

The new object could also be assigned to another variable or processed in other ways.

6.3.3 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
```

6.3.2 Utilización de 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()
```

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));
         }
}
```

Igualdad / identidad

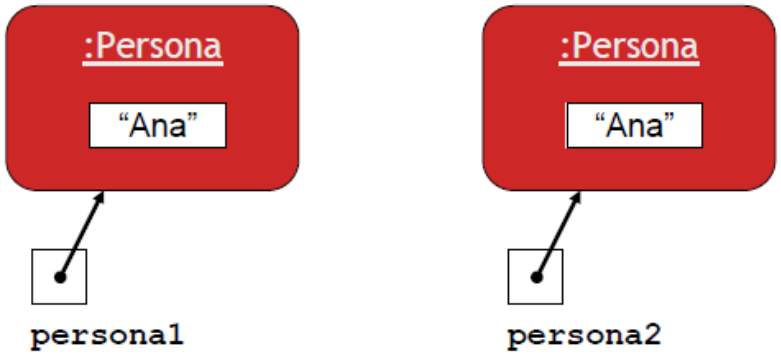
```
if(entrada == "hola") {
     ...
}
```

Prueba la identidad

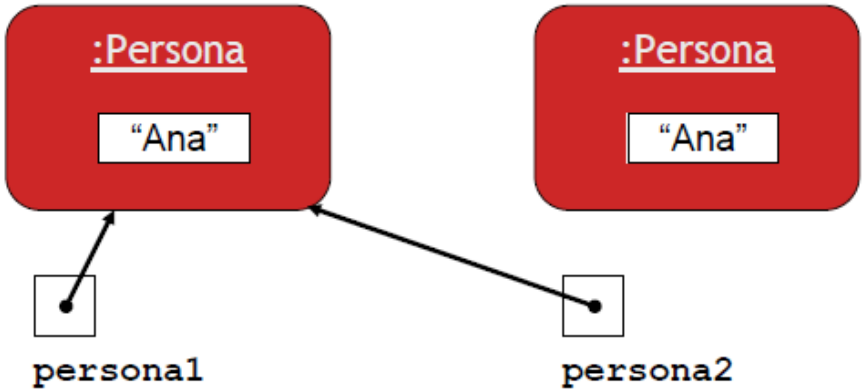
```
if(entrada.equals("hola")) {
     ...
}
```

Prueba la igualdad

- Todos los objetos Java se deben comparar siempre con `.equals`
- Los tipos básicos (incluidos referencias a objetos) se comparan con `==`




```
persona1 == persona2 ?
```




```
persona1 == persona2 ?
```

Número de Línea	Código
4	`String cadena1 = "Halo";`
5	`String cadena2 = "HALO";`
6	`cadena1.toUpperCase();`

¿Cuál sería el resultado de evaluar: `if (cadena1.equals(cadena2))`?

a. true
b. false
c. Error en la expresión
d. Ninguna de las anteriores es correcta

6.4 Adición de 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.

6.4.1 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`.

6.4.3 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?");
    }
}
```

6.5 Paquetes e importación

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.*;
```

6.6.1 Concepto de Mapa

- Esta Interface nos permite representar una estructura de datos para almacenar pares “clave/valor”.
  - Map<K,V>
- Un Map no puede tener claves duplicadas.
- La clave funciona como un identificador único.
- Las claves pueden ser de cualquier tipo de objetos.
- Si añadimos un nuevo elemento clave/valor cuando la clave ya existe, se sobrescribe el valor almacenado.
- Los más utilizados como clave son los objetos predefinidos de Java como:
  - **String.**
  - **Integer.**
  - **Double.**
- Los Map no permiten datos atómicos.

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

Los principales métodos para trabajar con los Map

- **nombreMap.size();**
  - Devuelve el número de elementos del Map.
- **nombreMap.isEmpty();**
  - Devuelve:
    - **true** si no hay elementos.
    - **false** si hay elementos.
- **nombreMap.put(K clave, V valor);**
  - Añade un elemento al Map.
- **nombreMap.get(K clave);**
  - Devuelve el **valor** asociado a la clave que se le pasa como parámetro o "**null**" si la clave no existe.
- **nombreMap.clear();**
  - Borra todos los componentes del Map.
- **nombreMap.remove(K clave);**
  - Borra el par clave/valor de la clave que se le pasa como parámetro.
- **nombreMap.containsKey(K clave);**
  - Devuelve true si en el map hay una clave que coincide con K.
- **nombreMap.containsValue(V valor);**
  - Devuelve true si en el map hay un Valor que coincide con V.
- **nombreMap.values();**
  - Devuelve una "Collection" con los valores del Map.

Clases de MAP

- HashMap
  - Los elementos que inserta en el Map no tendrán un orden específico.
  - No aceptan claves duplicadas ni valores nulos.
- TreeMap
  - El Map lo ordena de forma natural.
  - Si la clave son valores enteros, entonces los ordena de menor a mayor.
- LinkedHashMap
  - Inserta en el Map los elementos en el orden en el que se van insertando
  - No tiene una ordenación de los elementos como tal, por lo que esta clase realiza las búsquedas de los elementos de forma más lenta que las demás clases.

		Implementaciones				
		Tabla Hash	Array Dinámico	Árbol Balanceado	Listas Enlazadas	Tabla Hash Listas Enlazadas
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

6.6.2 Utilización de un HashMap

- Los elementos que inserta en el Map no tendrán un orden específico.
- No aceptan claves duplicadas.
- Los mapas tipo HashMap organizan los elementos en una tabla hash, proporciona una eficiencia constante a las operaciones de búsqueda e inserción.
- Ejemplos de método **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**.

Con un HashMap

```
Map<Integer, String> map = new HashMap<Integer, String>();
map.put(1, "Casillas");          map.put(15, "Ramos");
map.put(3, "Pique");             map.put(5, "Puyol");
map.put(11, "Capdevila");        map.put(14, "Xabi Alonso");
map.put(16, "Busquets");         map.put(8, "Xavi Hernandez");
map.put(18, "Pedrito");          map.put(6, "Iniesta");
map.put(7, "Villa");

// Imprimimos el Map con un Iterador
Iterator it = map.keySet().iterator();
while(it.hasNext()){
  Integer key = it.next();
  System.out.println("Clave: " + key + " -> Valor: " + map.get(key));
}
```

```
Clave: 16 -> Valor: Busquets
Clave: 1  -> Valor: Casillas
Clave: 18 -> Valor: Pedrito
Clave: 3  -> Valor: Pique
Clave: 5  -> Valor: Puyol
Clave: 6  -> Valor: Iniesta
Clave: 7  -> Valor: Villa
Clave: 8  -> Valor: Xavi Hernandez
Clave: 11 -> Valor: Capdevila
Clave: 14 -> Valor: Xabi Alonso
Clave: 15 -> Valor: Ramos
```

Con un TreeMap

```
Map<Integer, String> treeMap = new TreeMap<Integer, String>();
treeMap.put(1, "Casillas");     treeMap.put(15, "Ramos");
treeMap.put(3, "Pique");        treeMap.put(5, "Puyol");
treeMap.put(11, "Capdevila");   treeMap.put(14, "Xabi Alonso");
treeMap.put(16, "Busquets");    treeMap.put(8, "Xavi Hernandez");
treeMap.put(18, "Pedrito");     treeMap.put(6, "Iniesta");
treeMap.put(7, "Villa");

// Imprimimos el Map con un Iterador que ya hemos instanciado anteriormente
it = treeMap.keySet().iterator();
while(it.hasNext()){
  Integer key = it.next();
  System.out.println("Clave: " + key + " -> Valor: " + treeMap.get(key));
}
```

```
Clave: 1  -> Valor: Casillas
Clave: 3  -> Valor: Pique
Clave: 5  -> Valor: Puyol
Clave: 6  -> Valor: Iniesta
Clave: 7  -> Valor: Villa
Clave: 8  -> Valor: Xavi Hernandez
Clave: 11 -> Valor: Capdevila
Clave: 14 -> Valor: Xabi Alonso
Clave: 15 -> Valor: Ramos
Clave: 16 -> Valor: Busquets
Clave: 18 -> Valor: Pedrito
```

```
System.out.println("********* Trabajando con los métodos de Map *********");
System.out.println("Mostramos el numero de elementos que tiene el TreeMap: treeMap.size() = "+treeMap.size());
System.out.println("Vemos si el TreeMap esta vacio : treeMap.isEmpty() = "+treeMap.isEmpty());
System.out.println("Obtenemos un elemento del Map pasandole la clave 6: treeMap.get(6) = "+treeMap.get(6));
System.out.println("Borramos un elemento del Map el 18 (porque fue sustituido): treeMap.remove(18)"+treeMap.remove(18));
System.out.println("Vemos que pasa si queremos obtener la clave 18 que ya no existe: treeMap.get(18) = "+treeMap.get(18));
System.out.println("Vemos si existe un elemento con la clave 18: treeMap.containsKey(18) = "+treeMap.containsKey(18));
System.out.println("Vemos si existe un elemento con la clave 1: treeMap.containsKey(1) = "+treeMap.containsKey(1));
System.out.println("Vemos si existe el valo 'Villa' en el Map: treeMap.containsValue("Villa") = "+treeMap.containsValue("Vill
a"));
System.out.println("Vemos si existe el valo 'Ricardo' en el Map: treeMap.containsValue("Ricardo") = "+treeMap.containsValue("
Ricardo"));
System.out.println("Borramos todos los elementos del Map: treeMap.clear()");treeMap.clear();
System.out.println("Comprobamos si lo hemos eliminado viendo su tamaño: treeMap.size() = "+treeMap.size());
System.out.println("Lo comprobamos tambien viendo si esta vacio treeMap.isEmpty() = "+treeMap.isEmpty());
```

```
********* Trabajando con los métodos de Map *********
Mostramos el numero de elementos que tiene el TreeMap: treeMap.size() = 11
Vemos si el TreeMap esta vacio : treeMap.isEmpty() = false
Obtenemos un elemento del Map pasandole la clave 6: treeMap.get(6) = Iniesta
Borramos un elemento del Map el 18 (porque fue sustituido): treeMap.remove(18)Pedrito
Vemos que pasa si queremos obtener la clave 18 que ya no existe: treeMap.get(18) = null
Vemos si existe un elemento con la clave 18: treeMap.containsKey(18) = false
Vemos si existe un elemento con la clave 1: treeMap.containsKey(1) = true
Vemos si existe el valo 'Villa' en el Map: treeMap.containsValue("Villa") = true
Vemos si existe el valo 'Ricardo' en el Map: treeMap.containsValue("Ricardo") = false
Borramos todos los elementos del Map: treeMap.clear()
Comprobamos si lo hemos eliminado viendo su tamaño: treeMap.size() = 0
Lo comprobamos tambien viendo si esta vacio treeMap.isEmpty() = true
```

Pregunta: Dada la siguiente declaración:

```
Map < String, Double > map = new HashMap < String, Double > ();
```

¿Cuál de las siguientes opciones es correcta?

a. `map.add( " pi " , 3.14159);`
b. `map.add( " e " , 2.71828D);`
c. `map.add( " log(1) " , new Double(0.0));`
d. Ninguna de las anteriores

Lo orden es put no add

6.7 Utilización de conjuntos

Un **conjunto** es un 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`.

Pregunta: Dado el siguiente código:

```
30. Set < Object > objetos = new HashSet < Object > ();
31. String one = "hola";
32. int two = 2;
33. Boolean three = new Boolean(true);
34. objetos.add(one);
35. objetos.add(two);
36. objetos.add(three);
37. objetos.add(three);
38. for(Object objeto : objetos) {
39.    System.out.print(objeto);}
```

¿Cuál de las siguientes afirmaciones es cierta?

a. La salida es hola, 2 y true en un orden no determinado.

a. La salida es hola, 2, true y true en un orden no determinado.

b. Error de compilación en la línea 35.

c. Excepción en tiempo de ejecución en la línea 37.

Pregunta: Dado el siguiente código, ¿Cuál de las siguientes afirmaciones es correcta?

```
Set < Object > objetos = new HashSet<Object>();
String obj1 = "JAVA";
int obj2 = 5;
Boolean obj3 = new Boolean(true);
objetos.add(obj3);
objetos.add(obj1);
objetos.add(obj2);
objetos.add(obj3);
for(Object object : objetos) {
System.out.print(object);
```

a. Error en tiempo de ejecución.
b. Se muestran por pantalla *JAVA 5* y *true* en un orden no determinado.
c. Se muestran por pantalla *JAVA 5* y *true* en el orden exacto en el que fueron insertadas en la colección.
d. Se muestran por pantalla *JAVA 5* y *true* en un orden no determinado y, además, "true" se muestra dos veces.

Pregunta: Indique cuál de las siguientes afirmaciones es verdadera:

a. Para definir una variable de instancia es necesario utilizar la palabra reservada **static**.
b. Un método estático puede acceder a cualquier componente (método o variable) no estático de su clase.
c. Los métodos estáticos pueden ser sobreescritos.
d. Una variable de clase puede ser modificada sin necesidad de haber instanciado objeto alguno de dicha clase.

6.8 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.lineaSiguiente().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.

Pregunta: Dado el siguiente fragmento de código, indique cuál de las siguientes afirmaciones es el resultado de su ejecución:

```
if(" String ".trim() == "String")
    System.out.println("Igual");
else
    System.out.println("No Igual");
```

a. El código compilará e imprimirá "Igual".
b. El código compilará e imprimirá "No Igual".
c. El código provocará un error de compilación.
d. El código provocará un error en tiempo de ejecución.

Número de Línea	Código
4	`String s = new String ("Bicycle");`
5	`int iBegin = 1;`
6	`char iEnd = 3;`
7	`System.out.println(s.substring(iBegin,iEnd));`

a. Bic
b. ic
c. icy

fraccionDeString = nombreDelString.substring (carácter Inicial Incluido, carácter Final Excluido)

6.10 Autoboxing y clase envoltorio

Clases «envoltorio»

En Java, cada tipo primitivo tiene su correspondiente clase «envoltorio» que representa el mismo tipo pero que en realidad, es un tipo objeto. Estas clases hacen posible que se usen valores de tipos primitivos en los lugares en que se requieren tipos objeto mediante un proceso conocido como *autoboxing*. La siguiente tabla enumera los tipos primitivos y sus correspondientes clases envoltorio del paquete `java.lang`. Excepto `Integer` y `Character`, los nombres de las clases envoltorio coinciden con los nombres de los tipos primitivos, pero con su primera letra en mayúscula.

Tipo primitivo	Tipo envoltorio
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

```
int i = 18;
Integer iwrap = new Integer(i);
…
int value = iwrap.intValue();
```

Envuelve el valor

Lo desenvuelve

```
private ArrayList<Integer> markList;

…

public void storeMark(int mark)

{

    markList.add(mark);

}
```

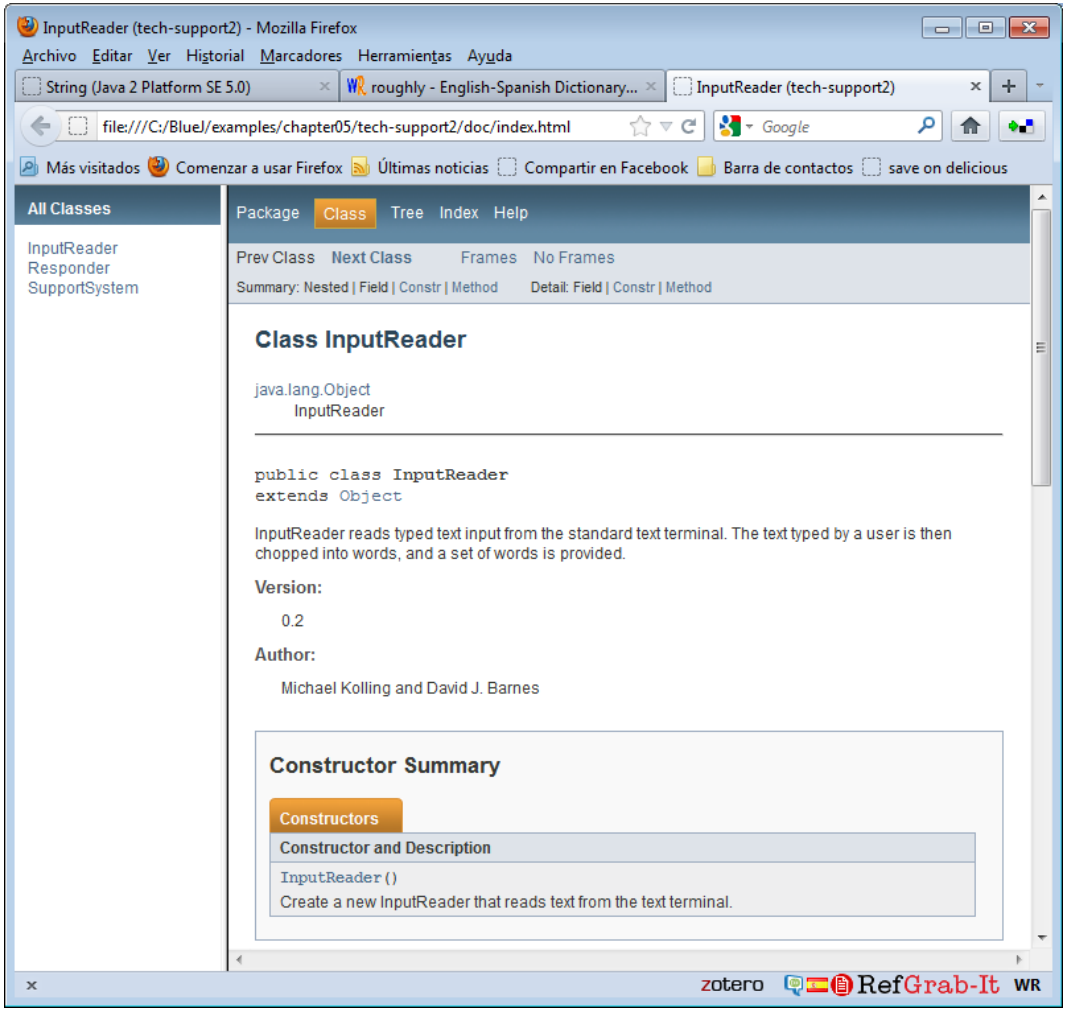
autoboxing

```
int firstMark = markList.remove(0);
```

unboxing

6.11 Escribir documentación de clase

- (BlueJ) Herramientas -> documentación proyecto



6.11.2 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.

6.12 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 signaturas 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.

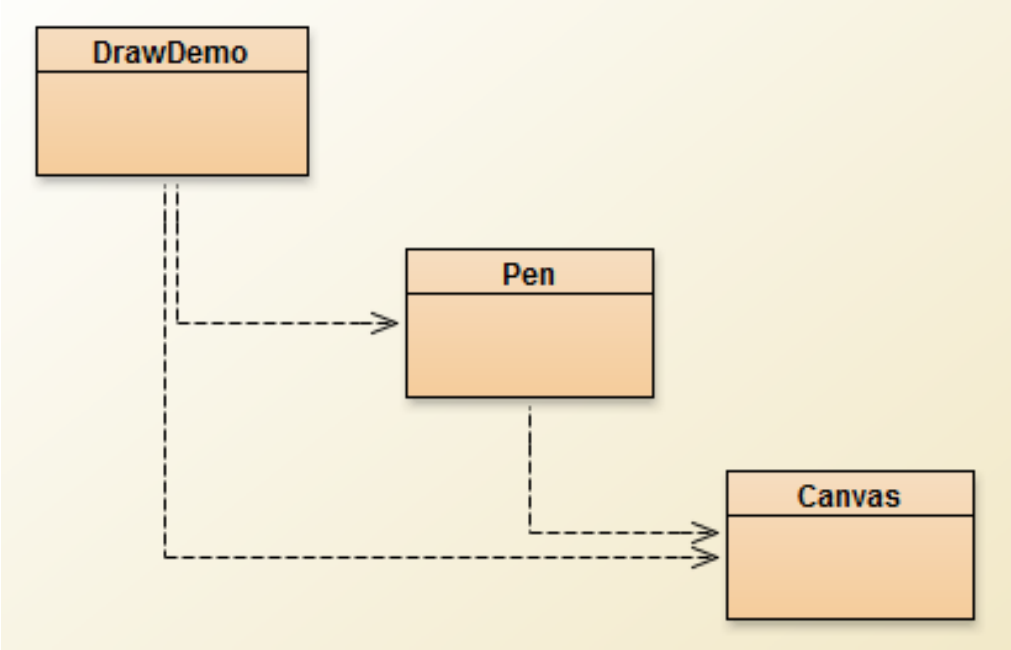
Visibilidad

- Usar **private** para métodos y variables
  - Que solamente se utilicen dentro de la clase y que deberían estar ocultas para todo el resto
- Usar **public** para métodos, constantes y otras variables importantes
  - Que deban ser visibles para todo el mundo
- Usar **protected**
  - Si se quiere que las clases del mismo paquete puedan tener acceso a estas variables o métodos
- No usar nada
  - Dejar la visibilidad por defecto (*default*, *package*) para métodos y variables que deban estar ocultas fuera del paquete, pero que deban estar disponibles al acceso desde dentro del mismo paquete.

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	✔		✔	

Visibilidad	public	protected	default	private
UML	+	#	~	–
Misma Clase	SI	SI	SI	SI
Cualquier clase del mismo paquete	SI	SI	SI	NO
Subclase mismo Paquete	SI	SI	SI	NO
Subclase diferente Paquete	SI	SI Herencia	NO	NO
Cualquier clase fuera del paquete	SI	NO	NO	NO

6.13 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.
    /**
     * Pre
     */
    public
    {
        my
        ra
    }

    /**
     * Dra
     */
    public
    {
        Pen pen = new Pen(320, 260, myCanvas);
        pen.setColor(Color.BLUE);
```

```
void        clear()
Object      clone()
void        colorScribble()
void        drawSquare()
void        drawWheel()
boolean     equals(Object)
void        finalize()
Class<?>    getClass()
int         hashCode()
void        notify()
void        notifyAll()
```

DrawDemo

`void clear()`

Clear the screen.

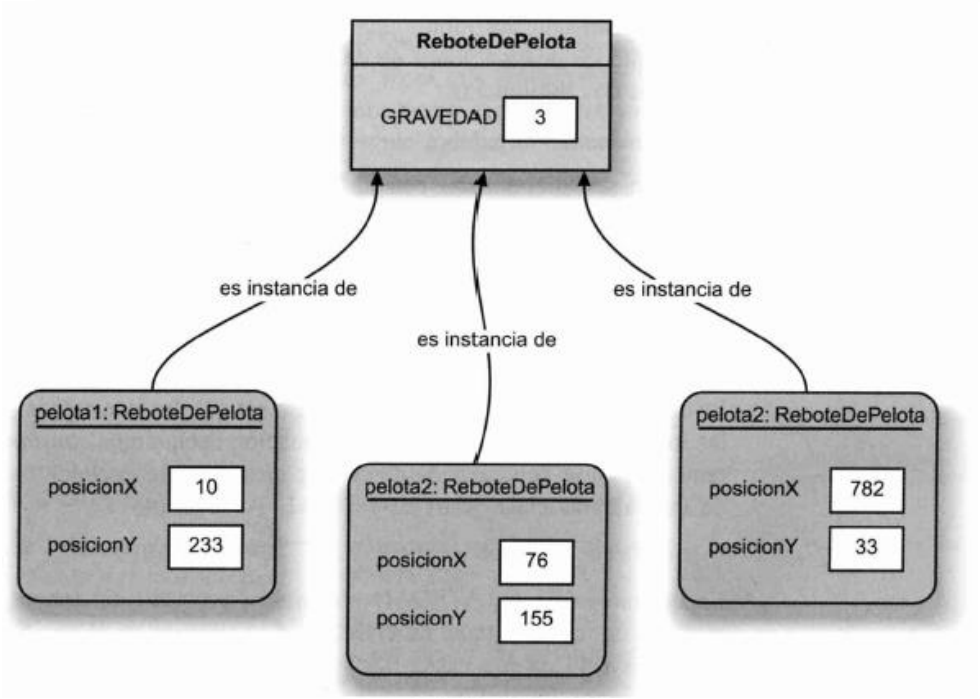
Ctrl + espacio

6.14 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 RebotePelota
{
   // Efecto de gravedad
   private static final int GRAVEDAD = 3;

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



6.14.2 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;
```

Costantes 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 *constantes 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
```

6.15 Ejecutar un programa fuera de Buej
Métodos de clase

Un método de clase se define agregando la palabra clave `static` antes del nombre del tipo en la signatura del método:

```
public static int getNumeroDeDiasDeEsteMes()
{
   ...
}
```

Estos métodos puede ser invocados utilizando la notación usual de punto, especificando el nombre de la clase en que está definido seguido del punto y luego del nombre del método. Si, por ejemplo, el método anterior está declarado en una clase de nombre `Calendario`, la siguiente sentencia lo invoca:

```
int dias = Calendario.getNumeroDeDiasDeEstemes();
```

Observe que antes del punto se usa el nombre de la clase, no se ha creado ningún objeto

Métodos y campos de clase

- Tienen estas limitaciones:
  - a) No pueden acceder a campos de instancia (lógico, pues los campos van asociados a objetos).
  - b) No pueden invocar a un método de instancia de la misma clase (lógico pues los métodos de instancia van asociados a objetos).
    - Los métodos de clases no pueden utilizar ni los campos ni los métodos de las instancias definidos en la clase
    - Un método de clase solo puede invocar a otros métodos de clases definidos en su propia clase

El método “main”

En Java, este problema se resuelve usando una convención: cuando se inicia un programa Java, el nombre de la clase se especifica como un parámetro del comando de inicio y el nombre del método es siempre el mismo, el nombre de este método es «main». Por ejemplo, considere el siguiente comando ingresado en una línea de comando, como si fuera un comando de Windows o de una terminal Unix:

```
java Juego
```

El comando `java` inicia la máquina virtual de Java, que forma parte del kit de desarrollo de Java (SDK) y que debe estar instalado en su sistema. `Juego` es el nombre de la clase que queremos iniciar.

Luego, el sistema Java buscará un método en la clase `Juego` cuya signatura coincida exactamente con la siguiente:

```
public static void main(String[] args)
```

Desarrollar fuera del BlueJ

Si no quiere solamente ejecutar programas, sino que también quiere desarrollarlos fuera del entorno BlueJ, necesitará editar y compilar las clases. El código de una clase se almacena en un archivo de extensión «.java»; por ejemplo, la clase `Juego` se almacena en un archivo de nombre `Juego.java`. Los archivos fuente pueden editarse con cualquier editor de textos. Existen muchos editores de textos libres o muy baratos. Algunos, como el *Notepad* o el *WordPad* se distribuyen con Windows, pero si en realidad quiere usar un editor para hacer algo más que una prueba rápida, querrá obtener uno mejor. Sin embargo, sea cuidadoso con los procesadores de texto: generalmente los procesadores de texto no graban en formato de texto plano y Java no podrá leerlos.

Los archivos fuente pueden compilarse desde una línea de comando usando el compilador Java que se incluye en el JDK y que se invoca mediante el comando `javac`. Para compilar un archivo fuente de nombre `Juego.java` use el comando

```
javac Juego.java
```

Este comando compilará la clase `Juego` y cualquier otra clase que dependa de ella; creará un archivo denominado `Juego.class` que contiene el código que puede ser ejecutado mediante la máquina virtual de Java. Para ejecutar este archivo use el comando

```
java Juego
```

Observe que este comando no incluye la extensión del archivo «.class».

Pregunta: Según el texto de la bibliografía básica de la asignatura, indique cual de las siguientes afirmaciones es correcta:

a. Un mapa es una colección que almacena pares llave/valor como entradas.
b. Un mapa es una colección que almacena tríos llave/índice/valor como entradas.
c. Un mapa es una colección que almacena pares índice/valor como entradas.
d. Un mapa es una colección que almacena tríos índice/posición/valor como entradas.

Pregunta: Dado el siguiente fragmento de código, indique cuál de las siguientes afirmaciones es el resultado de su ejecución:

```
public class TestSet {
   static void add(Set set) {
      set.add("Hola");
      set.add(1);
      System.out.println(set.size());
   }
   public static void main(String[] args) {
      Set<String> set = new HashSet<String>();
      add(set);
   }
}
```

a. 0
b. 1
c. 2
d. NullPointerException

Pregunta: Un conjunto es una estructura:

a. Que almacena cada elemento individual una sola vez como mínimo. No mantiene un orden específico.
b. Que almacena cada elemento individual una sola vez como mínimo. Mantiene un orden específico.
c. Que almacena cada elemento individual una sola vez como máximo. No mantiene un orden específico.
d. Que almacena cada elemento individual una sola vez como máximo. Mantiene un orden específico.

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.

Vídeos

- Colecciones
  - https://www.youtube.com/watch?v=bTu-fz1JmWQ&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk&index=179
  - https://www.youtube.com/watch?v=rqHBXAZ9F9k&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk&index=180
- Desde el capítulo 180 a 189