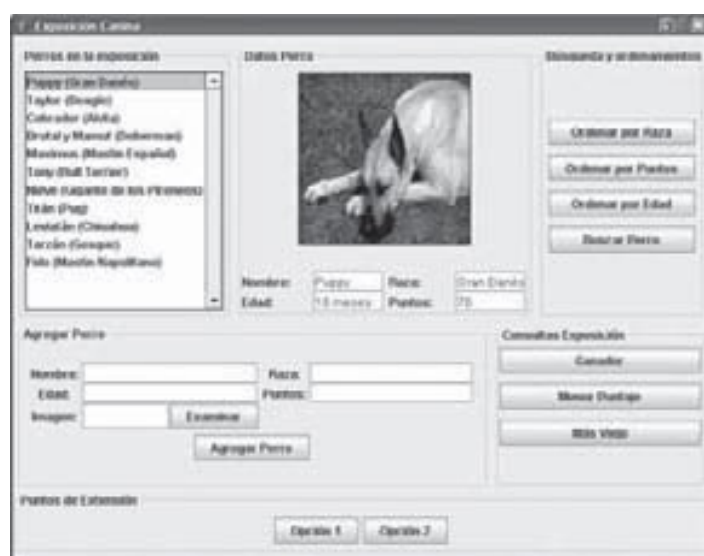


5. Caso de Estudio N° 3: Una Exposición Canina

En el último caso de este nivel vamos a construir un programa para manejar la información de una exposición canina. De cada uno de los perros que participa en la exposición nos interesa registrar su nombre (el cual debe ser único en toda la exposición), su raza, su edad en meses, una imagen y el número de puntos que le asignó el jurado.

En la figura 1.15 aparece la interfaz de usuario que se espera tenga el programa. Éste debe ofrecer los siguientes servicios al usuario: (1) mostrar la lista de los perros registrados en la exposición, ordenada por raza, puntos o edad; (2) mostrar la información de un perro específico, (3) registrar un nuevo perro, (4) localizar un perro por su nombre, (5) buscar al perro ganador de la exposición (el que tiene un mayor puntaje asignado), (6) buscar al perro con menor puntaje y (7) buscar al perro más viejo de todos (con mayor edad).

Fig. 1.15 – Interfaz de usuario para el caso de estudio de la exposición canina



- En la parte izquierda de la ventana aparece la lista de perros registrados en la exposición. Inicialmente no están ordenados.
- Usando los botones que aparecen en la parte derecha de la ventana, se puede ordenar esta lista por distintos conceptos.
- En la parte central aparece la información del perro que se encuentra seleccionado en la lista.
- Para registrar un nuevo perro se deben ingresar sus datos y oprimir el botón **Agregar Perro**.
- Todos los requerimientos de búsqueda aparecen asociados con alguno de los botones de la interfaz.

El programa va a manejar un esquema muy simple de persistencia, en el cual el estado inicial del programa debe cargarse desde un archivo de propiedades

llamado "perros.txt", localizado en el directorio **data**, el cual tiene el formato que se ilustra en el siguiente cuadro:





```
total.perros = 2

perro1.nombre = Puppy
perro1.raza = Gran Danés
perro1.imagen = ../data//gran_danes.jpg
perro1.puntos = 70
perro1.edad = 10

perro2.nombre = Tarzán
perro2.raza = Gosque
perro2.imagen = ../data//gosque.jpg
perro2.puntos = 100
perro2.edad = 137
```

- La primera propiedad del archivo ("total.perros") define el número de perros presentes en el archivo.
- Para cada uno de los perros tenemos cinco datos: nombre, raza, imagen, puntos y edad.
- Cada uno de los datos de un perro se encuentra asociado con una propiedad.

5.1. Objetivos de Aprendizaje

¿Qué tenemos que hacer en el caso de estudio?	¿Qué tenemos que aprender o reforzar?
 Presentar en la interfaz de usuario la lista de los perros de la exposición.	 Estudiar el componente <code>JList</code> de Java, aprender a integrarlo a una interfaz de usuario y a tomar los eventos que sobre los elementos de la lista genere el usuario (clic sobre un perro).
 Ordenar por distintos conceptos un grupo de longitud variable de objetos.	 Adaptar lo visto en el caso anterior para hacer los ordenamientos pedidos, por distintos conceptos y teniendo en cuenta que vamos a manejar objetos en lugar de tipos simples.

5.2. Comprensión de los Requerimientos

Tarea 18



Objetivo: Entender el problema del caso de estudio de la exposición canina.

(1) Lea detenidamente el enunciado del caso de estudio de la exposición canina e (2) identifique y complete la documentación de los requerimientos funcionales que allí aparecen.

Requerimiento funcional 1	Nombre	R1 – Mostrar la lista de perros de la exposición
	Resumen	
	Entradas	
	Resultado	
Requerimiento funcional 2	Nombre	R2 – Mostrar la información de un perro
	Resumen	
	Entradas	

Requerimiento funcional 2	Resultado	
Requerimiento funcional 3	Nombre	R3 – Registrar un nuevo perro
	Resumen	
	Entradas	
	Resultado	
Requerimiento funcional 4	Nombre	R4 – Localizar un perro por su nombre
	Resumen	
	Entradas	
	Resultado	
Requerimiento funcional 5	Nombre	R5 – Buscar al perro ganador de la exposición
	Resumen	
	Entradas	
	Resultado	

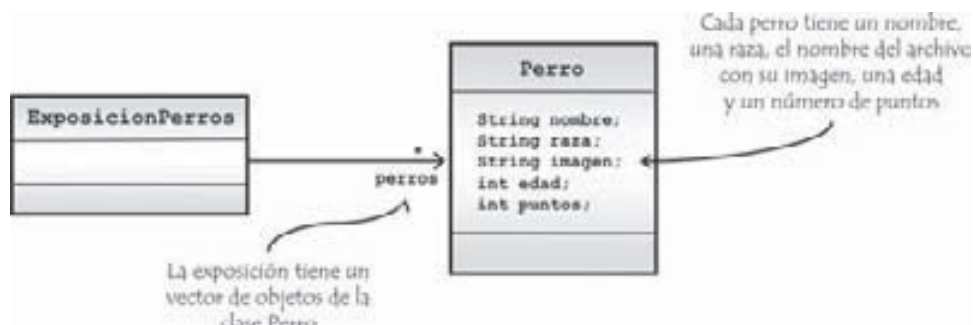
Requerimiento funcional 6	Nombre	R6 – Buscar al perro con menor puntaje
	Resumen	
	Entradas	
	Resultado	
Requerimiento funcional 7	Nombre	R7 – Buscar al perro más viejo
	Resumen	
	Entradas	
	Resultado	

5.3. Arquitectura de la Solución

El mundo del problema de este caso es muy simple, como se muestra en la figura 1.16. Sólo hay dos en-

tidades: la primera, que representa un perro (clase `Perro`), con todas sus características, y la segunda, la exposición (clase `ExposicionPerros`), que tiene un vector de perros como su única asociación.

Fig. 1.16 – Diagrama de clases del modelo del mundo



Los invariantes de estas clases son:

- Clase `ExposicionPerros`:
El vector de perros está inicializado (`perros != null`)
No hay dos perros en el vector que tengan el mismo nombre.
- Clase `Perro`:
El nombre del perro es una cadena no nula de caracteres (`nombre != null`)
La raza del perro es una cadena no nula de caracteres (`raza != null`)

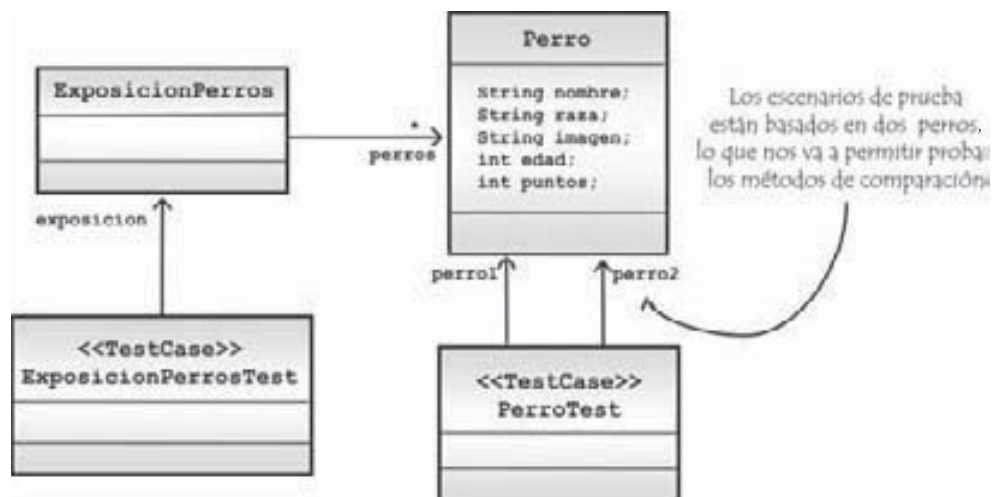
La imagen del perro (nombre del archivo) está dada por una cadena no nula de caracteres (`imagen != null`)

La edad del perro es un valor positivo (`edad > 0`)

El perro tiene un número no negativo de puntos (`puntos >= 0`)

Para las pruebas tendremos dos clases, como se muestra en la figura 1.17: una para probar la clase `Perro` y otra para probar la clase `ExposicionPerros`. El tema de las pruebas no será abordado en este caso de estudio, por lo que se recomienda revisar el contenido del CD para estudiar allí los escenarios utilizados y los casos de prueba definidos.

Fig. 1.17 – Diagrama de clases de las pruebas



5.4. Comparación de Objetos por Múltiples Criterios

Tanto los algoritmos de ordenamiento como los algoritmos de búsqueda se basan en la capacidad que ellos deben tener de comparar dos elementos y decidir si son iguales, mayores o menores. Sin eso no podrían trabajar. En el caso de las muestras, en las cuales teníamos valores de tipo simple (enteros), utilizamos los operadores relacionales (`==`, `<`, `>`, `<=`, `>=`, `!=`) que provee Java para tal fin. ¿Cómo hacer

ahora que queremos utilizar los mismos algoritmos, pero aplicados a objetos? ¿Cómo saber si un perro es mayor que otro cuando se quieren ordenar por raza?

Comencemos tratando el problema de la igualdad de objetos. Aquí tenemos dos niveles posibles: el primero se refiere al caso en el cual dos objetos son físicamente el mismo. Si eso es lo que queremos establecer, podemos utilizar el operador `==` de Java. Al decir `obj1 == obj2` estamos preguntando si las

variables `obj1` y `obj2` están haciendo referencia al mismo objeto en memoria. El segundo caso posible es cuándo queremos saber si dos objetos son iguales bajo algún concepto (por ejemplo, si dos perros tienen la misma raza). En ese caso, es necesario escribir un método llamado `equals()`, que indique si dos objetos son iguales sin ser necesariamente el mismo. Así es, por ejemplo, como comparamos dos cadenas de caracteres, ya que la clase `String` define este método. Al decir `cad1.equals(cad2)` estamos tratando de establecer si el "contenido" de `cad1` es igual al "contenido" de `cad2`, aunque sean objetos distintos.

El problema con el que nos encontramos ahora es que el método `equals()` sólo contempla un criterio de orden, y eso podría no ser suficiente en algunos casos. La clase `String`, por ejemplo, definió otro método para verificar igualdad llamado `equalsIgnoreCase()`, que ignora si las letras están en mayúsculas o minúsculas. Lo importante, en cualquier solución que utilicemos, es que sea la misma clase la que implemente estos métodos, pues decidir si un objeto de la clase es igual a otro es su responsabilidad.

Si extendemos el problema a determinar si un objeto es menor o mayor que otro, podemos encontrar ideas de solución en la clase `String`, la cual cuenta con los siguientes métodos:

- `compareTo(cad)` : Compara dos cadenas lexicográficamente, retornando: (1) un valor negativo si el objeto cadena es menor que el parámetro, (2) cero si las dos cadenas son iguales o (3) un valor positivo si el objeto es mayor que el parámetro.
- `compareToIgnoreCase(cad)` : Funciona igual que el método anterior, pero ignora si las letras se encuentran en mayúsculas o minúsculas.

Nosotros vamos a utilizar el mismo enfoque, generalizando la idea de tener un método de comparación por cada criterio de orden que pueda tener un objeto. En el caso de la exposición canina, los perros

se deben saber comparar por nombre, por raza, por edad y por puntos. Para cada uno de esos criterios, la clase debe proveer el respectivo método de comparación. Es así como en la clase `Perro` vamos a encontrar los siguientes métodos:

- `compararPorNombre(perro2)` : Compara dos perros usando como criterio su nombre y retorna: (1) un valor negativo si el objeto es menor que el parámetro `perro2` (en este caso, si el nombre del perro es menor que el nombre del perro que llega como parámetro), (2) cero si los dos perros tienen el mismo nombre o (3) un valor positivo si el objeto es mayor que el parámetro `perro2`.
- `compararPorRaza(perro2)` : Compara dos perros usando como criterio su raza y retorna: (1) un valor negativo si el objeto es menor que el parámetro `perro2`, (2) cero si los dos perros tienen la misma raza o (3) un valor positivo si el objeto es mayor que el parámetro `perro2`.
- `compararPorEdad(perro2)` : Compara dos perros usando como criterio su edad y retorna: (1) un valor negativo si el objeto es menor que el parámetro `perro2`, (2) cero si los dos perros tienen la misma edad o (3) un valor positivo si el objeto es mayor que el parámetro `perro2`.
- `compararPorPuntos(perro2)` : Compara dos perros usando como criterio el número de puntos obtenidos en la exposición y retorna: (1) un valor negativo si el objeto es menor que el parámetro `perro2`, (2) cero si los dos perros tienen el mismo número de puntos o (3) un valor positivo si el objeto es mayor que el parámetro `perro2`.

Fijese que con sólo modificar estos métodos de comparación, podemos ordenar de manera ascendente o descendente un grupo de perros, sin necesidad de modificar el algoritmo de ordenamiento.




En el ejemplo 14 mostramos la implementación de los métodos de comparación para el programa de la exposición canina.

Ejemplo 14



Objetivo: Escribir los métodos de comparación de una clase usando distintos criterios.

En este ejemplo implementamos los cuatro métodos de comparación que necesita la clase `Perro`, cuya especificación se dio anteriormente.



```
public int compararPorNombre( Perro p )
{
    return nombre.compareToIgnoreCase( p.nombre );
}
```

-  Este método compara dos perros usando como criterio su nombre.
-  La solución consiste en usar los nombres de los dos perros y delegar esta responsabilidad al método `compareToIgnoreCase()` de la clase `String`.
-  Puesto que el parámetro "p" pertenece a la misma clase en la cual estamos definiendo el método (clase `Perro`), tenemos derecho a hacer referencia de manera directa a sus atributos. En este caso, el atributo "nombre" contiene el nombre del perro sobre el cual se hace la invocación del método y el atributo "p.nombre" hace referencia al nombre del perro que llega como parámetro.



```
public int compararPorRaza( Perro p )
{
    return raza.compareToIgnoreCase( p.raza );
}
```

-  Este método compara dos perros usando como criterio su raza.
-  Utilizamos el mismo tipo de solución del método anterior, pero usando el atributo que contiene la raza de cada uno de los perros.

```
public int compararPorPuntos( Perro p )
{
    if( puntos == p.puntos )
        return 0;
    else if( puntos > p.puntos )
        return 1;
    else
        return -1;
}
```

-  Este método compara dos perros usando como criterio el número de puntos conseguidos en la exposición.
-  Si son iguales retorna 0, si el perro tiene más puntos que el parámetro p retorna 1, en cualquier otro caso retorna -1.

```
public int compararPorEdad( Perro p )
{
    if( edad == p.edad )
        return 0;
    else if( edad > p.edad )
        return 1;
    else
        return -1;
}
```

-  Este método compara dos perros usando como criterio su edad.
-  Si tienen la misma edad retorna 0, si el perro es más viejo que el parámetro p retorna 1, en cualquier otro caso retorna -1.

5.5. Métodos de Ordenamiento y Búsqueda de Objetos

En esta sección nos vamos a dedicar a adaptar los métodos de ordenamiento y búsqueda que vimos en el caso anterior para que puedan trabajar sobre objetos y puedan manejar distintos criterios de

ordenamiento. En el ejemplo 15 mostraremos la adaptación de la técnica de ordenamiento por inserción, aplicada al problema de ordenar los perros por puntaje. Luego plantearemos en términos de tareas al lector la adaptación de los demás algoritmos de ordenamiento y el algoritmo de búsqueda binaria.

Ejemplo 15



Objetivo: Mostrar la adaptación del algoritmo de ordenamiento por inserción al caso en el cual se deban manejar objetos.

En este ejemplo se presenta el método de la clase `ExposicionPerros` que ordena un vector de objetos en el orden definido por el método `compararPorPuntos()` de la clase `Perro`.

```
public void ordenarPorPuntos( )
{
    for( int i = 1; i < perros.size( ); i++ )
    {
        Perro porInsertar = ( Perro )perros.get( i );
        boolean termino = false;

        for( int j = i; j > 0 && !termino; j-- )
        {
            Perro actual = ( Perro )perros.get( j - 1 );

            if( actual.compararPorPuntos( porInsertar ) > 0 )
            {
                perros.set( j, actual );
                perros.set( j - 1, porInsertar );
            }
            else
                termino = true;
        }
    }
    verificarInvariante( );
}
```



Al tratar de adaptar el algoritmo de ordenamiento por inserción que escribimos antes, vamos a encontrar tres dificultades.



La primera es que la sintaxis para recuperar un elemento es más pesada, por lo que es conveniente agregar variables temporales ("porInsertar" y "actual").



La segunda es que no podemos asignar directamente los valores; luego debemos pasar por el método `set()` de la clase `ArrayList`.



La tercera es que no podemos comparar los objetos con el operador relacional `==`, lo que nos obliga a utilizar el método `compararPorPuntos()`.



Del resto, la traducción es directa.

Tarea 19

Objetivo: Adaptar los métodos de ordenamiento y búsqueda binaria al caso en el cual deben manipular objetos

Escriba los métodos de la clase `ExposicionPerros` que se piden a continuación.

1. Implemente el método que ordena por raza el vector de perros, usando la técnica de intercambio (burbuja).

```
public void ordenarPorRaza( )
{

}

}
```

2. Implemente el método que ordena por edad el vector de perros, usando la técnica de selección.

```
public void ordenarPorEdad( )
{

}

}
```

3. Implemente el método que hace búsqueda binaria por nombre en el vector de perros y retorna la posición en el vector en el que se encuentra o -1 si no hay ningún perro con ese nombre.

```
public int buscarBinarioPorNombre( String nombre )
{

}

}
```

5.6. Manejo de Grupos de Valores en la Interfaz de Usuario

En las interfaces de usuario más simples se utilizan componentes gráficos para mostrar a la persona información individual. Estos componentes corresponden principalmente a etiquetas y zonas de texto. En algunos casos, como en el de la exposición canina, necesitamos utilizar componentes que permitan visualizar grupos de elementos, ya sea para pedirle al usuario que seleccione algo (el perro del cual desea información) o simplemente para mos-

trarle un conjunto de datos (la lista de perros de la exposición).

En esta sección vamos a estudiar dos componentes gráficos que, manejados en conjunto, nos van a permitir manejar listas de elementos: la clase `JList` y la clase `JScrollPane`. Vamos a presentar el tema contestando cuatro preguntas y utilizando como ejemplo el código de la interfaz de usuario del caso de estudio.

- ¿Cómo crear y configurar una lista gráfica?

```
public class PanelListaPerros extends JPanel
{
    // -----
    // Atributos
    // -----

    private JList listaPerros;

    private JScrollPane scroll;
```



Para ilustrar la declaración, creación y configuración de la lista gráfica, mostraremos el código del panel que lo contiene.



Al igual que con los demás componentes gráficos, debemos declarar un atributo por cada uno que queramos incluir en la interfaz.



Declaramos una lista gráfica (`listaPerros`) y un componente de desplazamiento (`scroll`).