Ordenamiento de arreglos

Ordenamiento de arreglos

- Un arreglo de strings se puede ordenar por su *orden natural* (A-Z) usando el método Arrays.sort.

```
String[] p = {“Mercurio”, “Venus”, “Tierra”, “Marte”, “Júpiter”, “Saturno”, “Urano”, “Neptuno”};
Arrays.sort(p);
System.out.println(Arrays.toString(p)); // [Júpiter, Marte, Mercurio, Neptuno, Saturno, Tierra, Urano, Venus]
```

- Se puede ordenar en orden inverso (Z-A) usando, como segundo argumento a Array.sort, el método Collections.reverseOrder.

Ordenamiento de arreglos

```
Arrays.sort(p, Collections.reverseOrder());
System.out.println(Arrays.toString(p));  // [Venus, Urano, Tierra, Saturno, Neptuno, Mercurio, Marte, Júpiter]
```

Ordenamiento de arreglos

- Un arreglo de primitivos también se puede ordenar por su orden natural usando el método Arrays.sort.

```
int[] a = {17, 15, 22, 9, -4};
Arrays.sort(a);
System.out.println(Arrays.toString(a));     // -4, 9, 15, 17, 22
```

- Pero **no** se puede ordenar en orden inverso usando el método Collections.reverseOrder.

Ordenamiento de arreglos

Arrays.sort(a, Collections.reverseOrder());

Required type: T[]

Provided: int[]

reason: no instance(s) of type variable(s) T exist so that int[] conforms to T[]

- Dos soluciones:

1. Declarar un arreglo de objetos:

Integer[] a = {17, 15, 22, 9, -4};

Ordenamiento de arreglos

2. Ordenar el arreglo con Arrays.sort y luego invertirlo:

```
int[] a = {17, 15, 22, 9, -4};
Arrays.sort(a);
System.out.println(Arrays.toString(a));    // -4, 9, 15, 17, 22
reverseArray(a);
System.out.println(Arrays.toString(a));    // 22, 17, 15, 9, -4
```

- Hay que programar el método reverseArray.

Invertir un arreglo

```
// Intercambia el primer elemento con el último, el segundo con el penúltimo, …
public static void reverseArray(int[] array)
{
    int temp, size = array.length;
    for (int i = 0; i < size / 2; i++) {
        temp = array[i];
        array[i] = array[size - i - 1];
        array[size - i - 1] = temp;
    }
}
```

Ordenamiento de objetos

- Suponer que se tiene definida una clase Person:

```
public class Person {
    private final int id;
    private final String firstName;
    …
```

- Y que se declaró un arreglo, llamado people, de elementos Person.

```
Person[] people = new Person[10];
people[0] = new Person(…);
…
```

Ordenamiento de objetos

- La instrucción:

  Arrays.sort(people);

- Genera la excepción:

  Exception in thread "main" java.lang.ClassCastException: class Person cannot be cast to class java.lang.Comparable …

  etc…

- El motivo es que Java no sabe ordenar objetos de tipo Person.
- Es decir, no puede decidir si un objeto de tipo Person va antes o después que otro objeto de tipo Person.

Ordenamiento de objetos

- ¿Porqué Arrays.sort si funciona con arreglo de tipo String?
- Porque la clase String implementa la interface Comparable.
- Esta interface impone un ordenamiento total a los objetos de cada clase que la implementa.
- Este orden se conoce como el *orden natural* de la clase.
- La interface Comparable define un solo método:

  int compareTo(T o)

- El método compareTo de una clase define su *método de comparación natural*.

Método compareTo

- Dados dos objetos, p1 y p2, de tipo T.
- La llamada p1.compareTo(p2) regresa:

a) Un valor < 0 si p1 va antes que p2 en su orden natural.
b) Un valor > 0 si p1 va después que p2 en su orden orden natural.
c) 0 si p1 y p2 en otro caso.

Método compareTo

- Ejemplo con strings:

```
String s1 = “hola”;
String s2 = “mundo”;
s1.compareTo(s2);              // regresa < 0 porque s1 < s2
s2.compareTo(s1);              // regresa > 0 porque s2 > s1
s1.compareTo(“hola”);          // regresa 0 porque son el mismo string
if (s1.compareTo(s2) > 0) {    // regresa false
    …
```

Clases que implementan Comparable

Clase	Orden natural
Byte	Numérico con signo
Character	Numérico sin signo
Long	Numérico con signo
Integer	Numérico con signo
Short	Numérico con signo
Double	Numérico con signo
Float	Numérico con signo
BigInteger	Numérico con signo
BigDecimal	Numérico con signo
Boolean	Boolean.FALSE < Boolean.TRUE

Clases que implementan Comparable

Clase	Orden natural
File	Lexicográfico por nombre del path
String	Lexicográfico
Date	Cronológico
Instant, LocalDate, LocalTime, LocalDateTime, ZonedDateTime	Cronológico

Ejemplo 1

- Agregar el método compareTo a la clase Person para ordenar el arreglo people usando el campo id.

Ejemplo 1

```
public class Person implements Comparable<Person> {
    private final int id;
    private final String firstName;
    …
    public int compareTo(Person p)
    {
        return this.getId() - p.getId();
    }
```

Ejemplo 1

- Suponer que se desea ordenar el arreglo por el campo firstName.
- Se puede tomar ventaja que la clase String ya implementa Comparable y tiene definido el método compareTo.

Ejemplo 1

```
public class Person implements Comparable<Person> {
    private final int id;
    private final String firstName;
    …
    public int compareTo(Person p)
    {
        return this.getFirstName().compareTo(p.getFirstName())
    }
```

Ejemplo 2

- Agregar el método compareTo a la clase Point3D.

```
public class Point3D {
    private int x, y, z;
    …
```

- La comparación es por las coordenadas x de dos puntos. Si son iguales, se compara por y. Si x y y son iguales, se compara por z.

Ejemplo 2

```
public class Point3D implements Comparable<Point3D> {
    private int x, y, z;
        …
    public int compareTo(Point3D other)
    {
        if (this.x < other.x) {
            return -1;      // menor x
        }
        else if (this.x > other.x) {
            return 1;       // mayor x
        }
```

Ejemplo 2

```
else if (this.y < other.y) {
    return -1;      // igual x, menor y
}
else if (this.y > other.y) {
    return 1;       // igual x, mayor y
}
else if (this.z < other.z) {
    return -1;      // igual x, igual y, menor z
}
else if (this.z > other.z) {
    return 1;       // igual x, igual y, mayor z
}
```

Ejemplo 2

```
    else {
        return 0;       // igual x, igual y, igual z
    }
}
```

Trucos de Comparable

- La resta produce el resultado correcto:

```
public int compareTo(Point3D other)
{
    if (this.x != other.x) {
        return this.x – other.x;          // diferente x
    }
    else if (this.y != other.y){
        return this.y – other.y;          // misma x, compara y
    }
    else {
        return this.z – other.z;          // misma x, misma y, compara z
    }
}
```

Trucos de Comparable

- Ejemplo: ordenar los objetos Point3D por su distancia al origen.
- La distancia es un valor de tipo double.
- El truco de la resta no funciona para variables double.
- La razón es que compareTo debe regresar un valor de tipo int.
- Se puede usar el método Math.signum() o Double.compare().

Ejemplo 3

```
public int compareTo(Point3D p)
{
    double d1 = Math.sqrt(x * x + y * y + z * z);
    double d2 = Math.sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
    return Double.compare(d1, d2);
}
```

Resumen

- Los arreglos de valores primitivos se pueden ordenar por su orden natural mediante el método Arrays.sort().
- No hay un método en Java para ordenar arreglos de primitivos en orden inverso.
- Los arreglos de objetos se pueden ordenar por su orden natural y de forma inversa, siempre y cuando la clase implemente la interface Comparable.
- Algunas clases de Java ya implementan a Comparable.