Algoritmos y Estructura de Datos:

Ordenación

Guillermo Román Díez (groman@fi.upm.es)
Lars-Åke Fredlund (lfredlund@fi.upm.es)

Universidad Politécnica de Madrid

Curso 2021/2022

Ordenación

- Es muy frecuente necesitar estructuras de datos que se encuentren ordenadas *de alguna manera*
  - ¿Recordais a algun algoritmo que hemos mencionado que necesita que todos los da¡tos son ordenados?

Ordenación

- Es muy frecuente necesitar estructuras de datos que se encuentren ordenadas *de alguna manera*
  - ¿Recordais a algun algoritmo que hemos mencionado que necesita que todos los da¡tos son ordenados?
  - Cuando se trata de tipos básicos hay un **orden total** entre sus elementos: $\ldots < -5 < -4 < -3 < \ldots < 4 < 5 < \ldots$

Ordenación

- Es muy frecuente necesitar estructuras de datos que se encuentren ordenadas *de alguna manera*
  - ¿Recordais a algun algoritmo que hemos mencionado que necesita que todos los da¡tos son ordenados?
  - Cuando se trata de tipos básicos hay un **orden total** entre sus elementos: $\ldots < -5 < -4 < -3 < \ldots < 4 < 5 < \ldots$

Pregunta

¿qué ocurre con dos objetos? ¿cómo sabemos si o1 $<$ o2?

Ordenación

- Es muy frecuente necesitar estructuras de datos que se encuentren ordenadas *de alguna manera*
  - ¿Recordais a algun algoritmo que hemos mencionado que necesita que todos los da¡tos son ordenados?
  - Cuando se trata de tipos básicos hay un **orden total** entre sus elementos: $\ldots < -5 < -4 < -3 < \ldots < 4 < 5 < \ldots$

Pregunta

¿qué ocurre con dos objetos? ¿cómo sabemos si o1 $<$ o2?

  - La comparación de punteros no es lo que buscamos (aunque algunos lenguajes lo permitan, p.e. `C` o `C++`)
  - El orden entre dos objetos distintos tiene que establecerlo **el programador**
  - El orden puede depender de los valores de un atributo, de varios, o de una combinación de sus valores

Iterfaces Comparable<T> y Comparator<T>

Pregunta

¿Cómo hacemos esto en Java?

Iterfaces Comparable<T> y Comparator<T>

Pregunta

¿Cómo hacemos esto en Java?

- Mediante los siguientes interfaces podemos definir ordenes totales entre objetos
- Interfaz `java.lang.Comparable`:

```
public interface Comparable<T> {
  public int compareTo(T t);
}
```

- Interfaz `java.util.Comparator`:

```
public interface Comparator<T> {
   public int compare(T t1, T t2);
}
```

Interfaz Comparable<T>

- La implementación de `Comparable<T>` por la clase `T` hace que ésta tenga que implementar el método `compareTo` para comparar dos objetos de tipo `T`
- La implementación del método fija el **orden** de los objetos de tipo `T`
- La llamada `t1.compareTo(t2)` donde `t1` y `t2` son de tipo `T`, `compareTo` devuelve un entero `res` tal que:
  - $res < 0$ si t1 es menor que t2
  - $res == 0$ si t1 es igual que t2
  - $res > 0$ si t1 es mayor que t2
- Debe ser consistente con `equals`, es decir:

  if `t1.equals(t2)` $\rightarrow$ `t1.compareTo(t2) == 0`

Interaz Comparable<T>

```
public class Alumno implements Comparable<Alumno> {
  private int dni;
  private String nombre;
  private String apellidos;
  ...
  public int compareTo(Alumno a) {
    return dni - a.getDni();
  }
  ...
}
```

Interfaz Comparator<T>

- La implementación del interfaz `Comparator<T>` implementan un método de comparación `compare` para objetos de tipo `T`
- La llamada `c.compare(t1,t2)` donde `t1` y `t2` son de tipo `T`, y `c` es de tipo `Comparator<T>`, `compare` devuelve un entero `res` tal que:
  - $res < 0$ si t1 es menor que t2
  - $res == 0$ si t1 es igual que t2
  - $res > 0$ si t1 es mayor que t2
- Debe ser consistente con `equals`, es decir:

  if `t1.equals(t2)` $\rightarrow$ `c.compare(t1,t2) == 0`

Objetos no comparables

```
public class AlumnoIncomparable {
  private int dni;
  private String nombre;
  private String apellidos;
  ...
}
```

Ejemplo Comparator<T>

- Si queremos comparar dos objetos de tipo `AlumnoIncomparable`, lo podemos hacer usando el siguiente comparador

```
public class ComparadorAlumnosDNI
        implements Comparator<AlumnoIncomparable> {

  public int compare(AlumnoIncomparable o1,
                     AlumnoIncomparable o2) {

    return o1.getDni() - o2.getDni();

  }
}
```

Pregunta

¿tiene sentido tener diferentes comparadores para el mismo objeto?

Comparator<T> a partir de Comparable<T>

- Si un objeto es de una clase que implementa el interfaz Comparable<T> entonces se puede crear un objeto Comparator<T> por defecto para dicha clase:

```
class DefaultComparator<T> implements Comparator<T> {
  public int compare(T a, T b) {
    if (a instanceof Comparable<?>)
      return ((Comparable<T>) a).compareTo(b);
    else
      throw new UnsupportedOperationException();
  }
}

class DefaultComparator<T extends Comparable<T>>
              implements Comparator<T> {
  public int compare(T o1, T o2) {
    return o1.compareTo(o2);
  }
}
```

Estructuras de datos "ordenadas"

- Hay dos formas de hacer una estructura de datos que utilice el "orden" entre sus elementos
  - Almacenar en la estructura elementos que implementen el interfaz `Comparable`
  - Utilizar `Comparator` para que lleve a cabo la comparación entre los elementos
    - ★ En este caso no es necesario que los elementos implementen `Comparable`

Ejemplo: insertar en una lista ordenada

```
static <E> void insert(E e, IndexedList<E> arr)
```

- ¿Como podemos expresar que el tipo `E` es "totalmente ordenado"?

Ejemplo: insertar en una lista ordenada

```
static <E> void insert(E e, IndexedList<E> arr)
```

- ¿Como podemos expresar que el tipo `E` es "totalmente ordenado"?
- Alternativo 1:

```
static <E> void insert1(E e, IndexedList<E> arr,
                        Comparator<E> cmp)
```

  - Mandamos el comparator como argumento extra

Ejemplo: insertar en una lista ordenada

```
static <E> void insert(E e, IndexedList<E> arr)
```

- ¿Como podemos expresar que el tipo `E` es "totalmente ordenado"?
- Alternativo 1:

```
static <E> void insert1(E e, IndexedList<E> arr,
                        Comparator<E> cmp)
```

  - Mandamos el comparator como argumento extra

- Alternativo 2:

```
static <E extends Comparable<E>> void
                  insert2(E e,IndexedList<E> arr)
```

  - Un requisito de que `E` tiene que implementar la interfaz `Comparable`