Java

Taller de Programación en Java

UNICEN

Java

Ejemplo: Una Tabla

Una Tabla

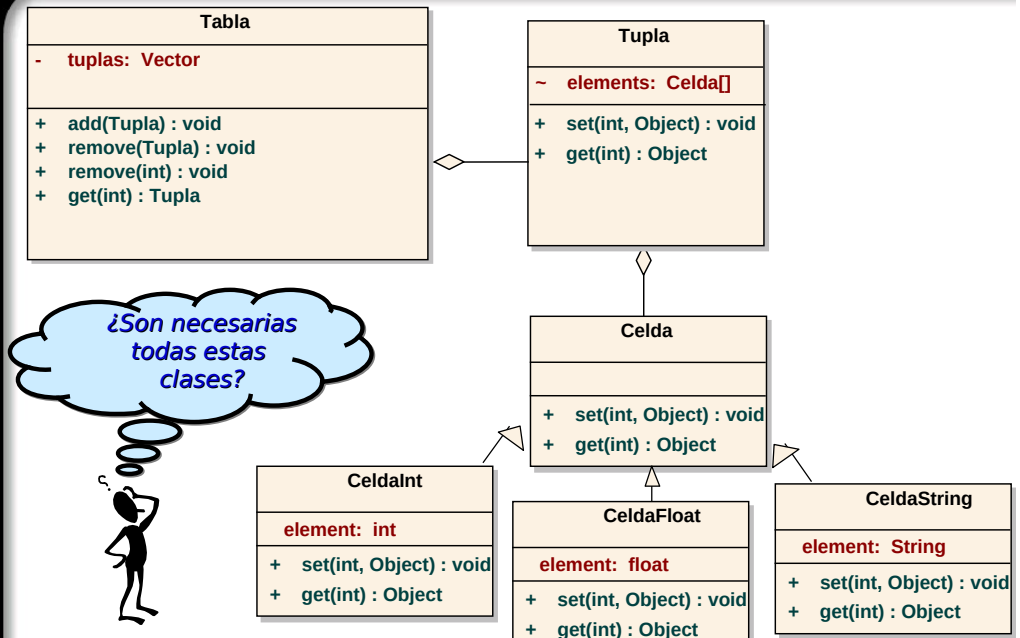
- Muchas son las aplicaciones que requieren organizar la información en una tabla, filtrarla y ordenarla:

Nombre / Tipo / Tupla

Número(Integer)	Nombre(String)	Sueldo(Float)	.....
1	Marcelo	1000	
4	Juan	1100	

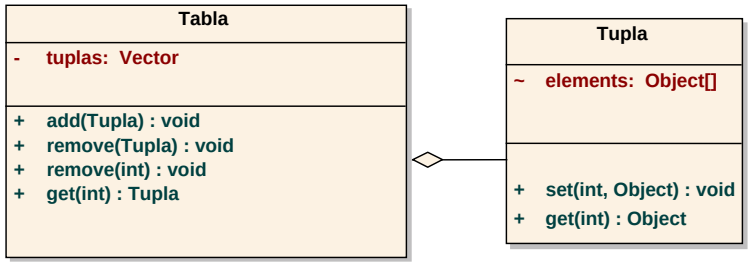
- Las tuplas se deben filtrar de acuerdo a distintos criterios(=, <, > , <>) o combinaciones lógicas de criterios (And, or, not)
  - ejemplos: *columna 2 = valor *(columna 1 = 4) and (columna 2 < 100)
- Las tuplas que satisfacen el filtro se deben ordenar teniendo en cuenta los valores de ciertas columnas de la tupla y con orden ascendente o descendente.
  - ejemplo: orden: primero por columna1 (ascendente), columna4 (descendente)
- El filtrado y el ordenamiento no se puede realizar sobre la misma tabla.
- No debe replicar la tabla.
- Provea además un mecanismo para recorrer una a una las tuplas.
- Verificación de tipos

La tabla



La tabla

No crear clases innecesarias.....



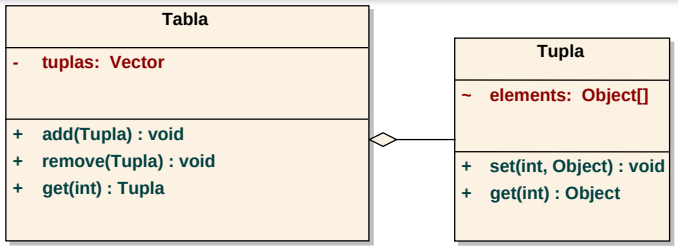
.... Hacer uso de los mecanismos que provee la orientacion a objetos (referencias polimorficas)

La Tabla

```
public class Tabla {

    private Vector tuplas;
    public Tabla() {
            tuplas =new Vector();  }

    public void add(Tupla t) {
            tuplas.add(t); }

    public void remove(Tupla t) {
       tuplas.remove(t);     }

    public void remove(int t) {
       tuplas.remove(t);  }

    public Tupla get(int i) {
       return (Tupla)tuplas.elementAt(i);
    }
```

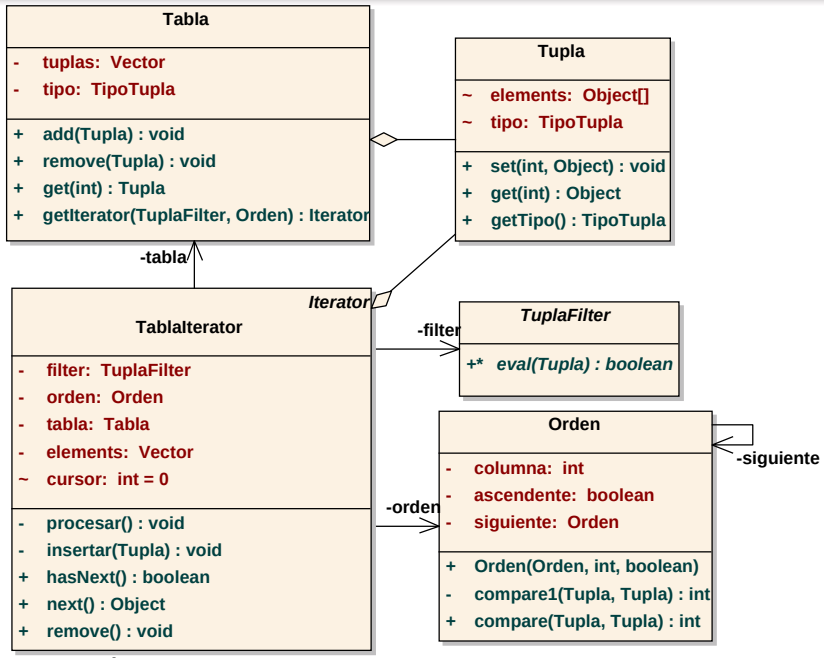
```
public class Tupla {

    Object[] elements;

    public Tupla(int t) {
            elements=new Object[t]; }

    public void set(int i,Object o) {
            elements[i]=o;  }

    public Object get(int i) {
       return elements[i];
    }
```


Iteración, filtrado y Ordenamiento



- El filtrado y el ordenamiento no se puede realizar sobre la misma tabla.
- No debe replicar la tabla.
- Provea además un mecanismo para recorrer una a una las tuplas.

Iteración, filtrado y Ordenamiento



Iteración

```
public class Tabla {
….
 public Iterator getIterator() {
       return tuplas.listIterator();    }

    public Iterator getIterator(TuplaFilter f,Orden o) {
       return new TablaIterator(this,f,o);    }
```

```
public class TablaIterator implements Iterator {

    private TuplaFilter filter;    private Orden orden;
    private Tabla tabla;
    private Vector elements;
    int cursor=0;

    public TablaIterator(Tabla t, TuplaFilter f,Orden o) {
       orden=o;        tabla=t;        filter=f;
       elements=new Vector();
       procesar();     }
```

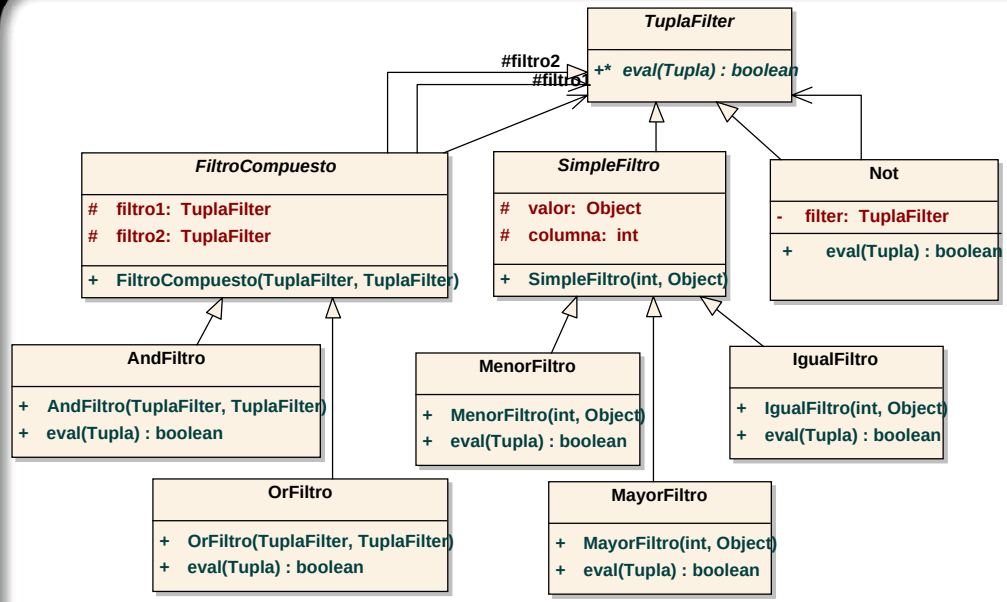

Iteración

```
private void procesar() {
   for (Iterator i=tabla.getIterator(); i.hasNext();) {
      Tupla t=(Tupla) i.next();
      if ( filter.eval(t)) insertar(t);
   }
}

private void insertar(Tupla t) {
   int i=0;
   while( i<elements.size() && (orden.compare(t,(Tupla)elements.elementAt(i))==1))
          i++;
   elements.insertElementAt(t, i);
}
```


Iteración

```
   public boolean hasNext() {
       return cursor < elements.size();
   }

   public Object next() {
       if (cursor < elements.size()) {
           return elements.elementAt(cursor++);
       } else return null;
   }

   public void remove() {
       if (cursor < elements.size()) {
           elements.removeElementAt(cursor);
       }
   }
```


Filtrado



Filtros Simples

```
public abstract class TuplaFilter {
    public abstract boolean eval(Tupla t);
}
```

```
public abstract class SimpleFiltro extends TuplaFilter {
      protected Object valor;
      protected int columna;
public SimpleFiltro(int columna, Object valor) {
      this.columna=columna;   this.valor=valor;    }   }
```

```
public class IgualFiltro extends SimpleFiltro {

  public IgualFiltro(int columna, Object valor) {
      super(columna, valor);    }

   public boolean eval(Tupla t) {
      return ((Comparable)t.get(columna)).compareTo(valor) == 0;  } }
```


Filtros Compuestos

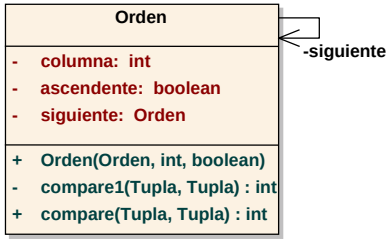
```
public abstract class TuplaFilter {
    public abstract boolean eval(Tupla t);
}
```

```
public abstract class FiltroCompuesto extends TuplaFilter {
   protected TuplaFilter filtro1,filtro2;
   public FiltroCompuesto(TuplaFilter f1, TuplaFilter f2) {
      filtro1=f1;        filtro2=f2;    }   }
```

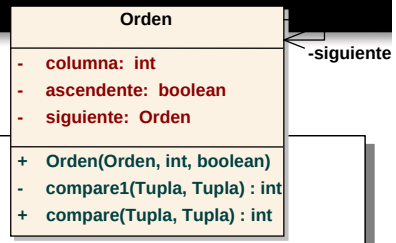
```
public class AndFiltro extends FiltroCompuesto {

   public AndFiltro(TuplaFilter f1,TuplaFilter f2) {
      super(f1,f2);    }

   public boolean eval(Tupla t) {
      return filtro1.eval(t) && filtro2.eval(t);   } }
```


Ordenamiento

```
public class Orden {

   private int columna;
   private boolean ascendente;
   private Orden siguiente;

   /** Creates a new instance of Orden */

   public Orden(Orden sig,int col,boolean asc) {
      siguiente=sig;
      columna=col;
      ascendente=asc;    }


   public int compare(Tupla t1,Tupla t2) {
      int r=compare1(t1,t2);
      if (r ==0) { if (siguiente != null) return siguiente.compare(t1,t2);
                else return 0;
      }  else return r;     } }
```



Ordenamiento




```
private int compare1(Tupla t1,Tupla t2) {
  if (ascendente) return
          ((Comparable) t1.get(columna)).compareTo(t2.get(columna));
 else return  ((Comparable) t2.get(columna)).compareTo(t1.get(columna));
}
```

```
private int compare1(Tupla t1,Tupla t2) {
return
          ((Comparable) t1.get(columna)).compareTo(t2.get(columna)) * orden;
                    //orden entero 1 o –1 según sentido
                    // se inicializa en el constructor }
```

Ordenamiento

```
public class ReverseComp implements Comparator {

    private Comparator comp;

    public ReverseComp(Comparator c) {
        comp=c;    }

    public int compare(Object obj, Object obj1) {
        return comp.compare(obj,obj1) * -1;      } }
```

Ordenamiento Reverso

```
for (Enumeration e=sis.listado(new ReverseComp(new ProductoComp()));
                                                    e.hasMoreElements();) {
        System.out.println(e.nextElement());
    }
```