Ejercicio U8_B3_E1:

```
interface Pila {
    //inserta un elemento en la cabeza de la pila
    void push(int dato);
    //saca un elemento de la cabeza de la pila.
    int pop();
    public boolean esVacia();
}

class MiPila implements Pila {

    private class Nodo {

        private Nodo sig;
        private int dato;

        public Nodo(int dato) {
            this.dato = dato;
            this.sig = null;
        }

        public Nodo(int dato, Nodo sig) {
            this.dato = dato;
            this.sig = sig;
        }


    }
    private Nodo cabeza = null;

    public void push(int dato) {
        if (cabeza == null) {
            cabeza = new Nodo(dato);
        } else {
            Nodo temp = new Nodo(dato, cabeza);
            cabeza = temp;
        }

    }

    public int pop() {
        int dato = cabeza.dato;
        cabeza = cabeza.sig;
        return dato;
    }

    public boolean esVacia() {
        return cabeza == null;
    }

}

class App {

    public static void main(String[] args) {
        MiPila mipila = new MiPila();
        mipila.push(1);
        mipila.push(2);
        mipila.push(3);
        mipila.push(4);
        mipila.push(5);
        while (!mipila.esVacia()) {
            System.out.println(mipila.pop());
        }
    }
}
```

Ejercicio U8_B3_E2:

Suponiendo que no es de interés general que la clase Articulo pueda ordenarse por código de artículo, pero si le interesa esa cuestión particular a la clase App, escribimos el comparador como clase interna.

Como interna static

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

class Articulo {
    String codArticulo;
    String descripcion;
    int cantidad;
    Articulo(String codArticulo, String descripcion, int cantidad) {
        this.codArticulo = codArticulo;
        this.descripcion = descripcion;
        this.cantidad = cantidad;
    }
}

class App {

    static class ComparadorArticulos implements Comparator<Articulo>{
        @Override
        public int compare( Articulo o1, Articulo o2) { return o1.codArticulo.compareTo(o2.codArticulo); }
    }

    public static void main(String[] args) {
        LinkedList<Articulo> articulos = new LinkedList<Articulo>();
        articulos.add(new Articulo("34","cuchillo",5));
        articulos.add(new Articulo("12","tenedor",7));
        articulos.add(new Articulo("41","cuchara",4));
        articulos.add(new Articulo("11","plato",6));

        Collections.sort(articulos, new App.ComparadorArticulos());
        for(Articulo a:articulos)
            System.out.println(a.codArticulo+", "+a.descripcion+", "+a.cantidad);
    }
}
```

Observa que Collections.sort(articulos, new App.ComparadorArticulos());
es equivalente como ya sabes a Collections.sort(articulos, new ComparadorArticulos()); pero esta segunda forma es más confusa.

Como interna pero no static

Tenemos que crear un objeto App()

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

class Articulo {
    String codArticulo;
    String descripcion;
    int cantidad;
    Articulo(String codArticulo, String descripcion, int cantidad) {
        this.codArticulo = codArticulo;
        this.descripcion = descripcion;
        this.cantidad = cantidad;
    }
}

class App {
```

```
    class ComparadorArticulos implements Comparator<Articulo>{
        @Override
        public int compare( Articulo o1, Articulo o2) { return o1.codArticulo.compareTo(o2.codArticulo); }
    }

    public static void main(String[] args) {
        App U8= new App();
        LinkedList<Articulo> articulos = new LinkedList<Articulo>();
        articulos.add(new Articulo("34","cuchillo",5));
        articulos.add(new Articulo("12","tenedor",7));
        articulos.add(new Articulo("41","cuchara",4));
        articulos.add(new Articulo("11","plato",6));

        Collections.sort(articulos, U8.new ComparadorArticulos());
        for(Articulo a:articulos)
            System.out.println(a.codArticulo+", "+a.descripcion+", "+a.cantidad);
    }
}
```

Como interna local (dentro de llaves de main)

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

class Articulo {

    String codArticulo;
    String descripcion;
    int cantidad;

    Articulo(String codArticulo, String descripcion, int cantidad) {
        this.codArticulo = codArticulo;
        this.descripcion = descripcion;
        this.cantidad = cantidad;
    }

}

class App {

    public static void main(String[] args) {
        class ComparadorArticulos implements Comparator<Articulo> {

            @Override
            public int compare(Articulo o1, Articulo o2) {
                return o1.codArticulo.compareTo(o2.codArticulo);
            }
        }

        LinkedList<Articulo> articulos = new LinkedList<Articulo>();
        articulos.add(new Articulo("34", "cuchillo", 5));
        articulos.add(new Articulo("12", "tenedor", 7));
        articulos.add(new Articulo("41", "cuchara", 4));
        articulos.add(new Articulo("11", "plato", 6));

        Collections.sort(articulos, new ComparadorArticulos());
        for (Articulo a : articulos) {
            System.out.println(a.codArticulo + ", " + a.descripcion + ", " + a.cantidad);
        }
    }
}
```

Ejercicio U8_B3_E3:

Como sólo necesitamos una instancia de la clase Comparador, este es un contexto apropiado para las clases internas anónimas.

Observa de paso como aquí si preciso utilizar el tipo, no me vale el operador diamante ya que si en lugar de new Comparator<Articulo>() escribo new Comparator() para sobreescribir compare() tengo argumentos de "tipo crudo" no Articulos

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

class Articulo {
   String codArticulo;
   String descripcion;
   int cantidad;
   Articulo(String codArticulo, String descripcion, int cantidad) {
      this.codArticulo = codArticulo;
      this.descripcion = descripcion;
      this.cantidad = cantidad;
   }
}

class App {

    public static void main(String[] args) {

      LinkedList<Articulo> articulos = new LinkedList<Articulo>();
      articulos.add(new Articulo("34","cuchillo",5));
      articulos.add(new Articulo("12","tenedor",7));
      articulos.add(new Articulo("41","cuchara",4));
      articulos.add(new Articulo("11","plato",6));

      Collections.sort(articulos, new Comparator<Articulo>(){
         @Override
          public int compare( Articulo o1, Articulo o2) { return o1.codArticulo.compareTo(o2.codArticulo); }
      });

      for(Articulo a:articulos)
         System.out.println(a.codArticulo+", "+a.descripcion+", "+a.cantidad);
   }
}
```

Ejercicio U8_B3_E4:

Recuerda el API del método Collections.sort

```
static <T> void                                  sort(List<T> list, Comparator<? super T> c)
                                                 Sorts the specified list according to the order induced by the specified comparator.
```

observa que ya que la lista es de artículos, el compilador infiere que el tipo del comparador es de artículos y de hecho observa como como las variables art1 y art2 se reconocen como artículos y desde netbeans puedo acceder a través de esas variables a las partes de un artículo

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

class Articulo {
   String codArticulo;
   String descripcion;
   int cantidad;
   Articulo(String codArticulo, String descripcion, int cantidad) {
      this.codArticulo = codArticulo;
      this.descripcion = descripcion;
      this.cantidad = cantidad;
   }
}

class App {
```

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

      LinkedList<Articulo> articulos = new LinkedList<Articulo>();
      articulos.add(new Articulo("34","cuchillo",5));
      articulos.add(new Articulo("12","tenedor",7));
      articulos.add(new Articulo("41","cuchara",4));
      articulos.add(new Articulo("11","plato",6));

      Collections.sort(articulos, (art1,art2) -> art1.codArticulo.compareTo(art2.codArticulo));


      for(Articulo a:articulos)
         System.out.println(a.codArticulo+", "+a.descripcion+", "+a.cantidad);
   }
}
```