Ejercicio U6_B7_E1:

```
import java.util.ArrayList;
import java.util.List;

class App{
    public static void main(String[] args){
        List<String> items = new ArrayList<>();
        items.add("A");
        items.add("B");
        items.add("C");
        items.add("D");
        items.add("E");

        items.stream()
                .filter(s->s.compareTo("B")>0)
                .forEach(s->System.out.println(s));

    }
}
```

Ejercicio U6_B7_E2:

```
import java.util.Arrays;
import java.util.List;

class App{
    public static void main(String[] args){
        List<String> myList =
                Arrays.asList("a1", "a2", "b1", "c2", "c1");

        Long l=myList
                .stream()
                .filter(s -> s.startsWith("c"))
                .count();

        System.out.println(l);

    }
}
```

Ejercicio U6_B7_E3:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

class Articulo {

    String codigo;
    double pvp;

    public Articulo(String codigo, double pvp) {
        this.codigo = codigo;
        this.pvp = pvp;
    }

    public String getCodigo() {
        return codigo;
    }

    public Double getPvp() {
        return pvp;
    }

    @Override
    public String toString() {
        return "Articulo{" + "codigo=" + codigo + ", pvp=" + pvp + '}';
```

```
   }


}

public class App {

   public static void main(String[] args) {
           Articulo[] articulos = {new Articulo("A1", 10.0), new Articulo("A2", 30.5), new Articulo("B1", 30.0), new
Articulo("B2", 100.0), new Articulo("c1", 66.5),};
      List<Articulo> listaArticulos = Arrays.asList(articulos);
      List<Articulo> filtrados;
      filtrados= listaArticulos
            .stream()
            .filter(a -> (a.pvp>30))
            .collect(Collectors.toList());
      System.out.println(filtrados);

   }
}
```

Ejercicio U6_B7_E4:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

class Articulo {

   String codigo;
   double pvp;

   public Articulo(String codigo, double pvp) {
      this.codigo = codigo;
      this.pvp = pvp;
   }

   public String getCodigo() {
      return codigo;
   }

   public Double getPvp() {
      return pvp;
   }

   @Override
   public String toString() {
      return "Articulo{" + "codigo=" + codigo + ", pvp=" + pvp + '}';
   }


}

public class App {

   public static void main(String[] args) {
           Articulo[] articulos = {new Articulo("A1", 10.0), new Articulo("A2", 30.5), new Articulo("B1", 30.0), new
Articulo("B2", 100.0), new Articulo("c1", 66.5),};
      List<Articulo> listaArticulos = Arrays.asList(articulos);
      List<Articulo> filtrados;
      filtrados= listaArticulos
            .stream()
            .filter(a -> (a.pvp>30))
            .sorted((a1,a2)->a1.getPvp()<=a2.getPvp()?-1:1)
            .collect(Collectors.toList());
      System.out.println(filtrados);

   }
}
```

con if en lugar de operador condicional hay que recordar que el cuerpo de la expresión lambda va entre {} y hay que tener cuidado con llaves y paréntesis

```
.sorted((a1, a2) -> {
               if (a1.getPvp() <= a2.getPvp()) {
```

```
                return -1;
            } else {
                return 1;
            }
        })
```

También podríamos invocar simplemente a sorted() si Articulo es comparable, por ejemplo

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

class Articulo implements Comparable<Articulo>{

    String codigo;
    double pvp;

    public Articulo(String codigo, double pvp) {
        this.codigo = codigo;
        this.pvp = pvp;
    }

    public String getCodigo() {
        return codigo;
    }

    public Double getPvp() {
        return pvp;
    }

    @Override
    public String toString() {
        return "Articulo{" + "codigo=" + codigo + ", pvp=" + pvp + '}';
    }

    @Override
    public int compareTo(Articulo o) {
        return new Double(pvp).compareTo(new Double(o.pvp));
    }




}

public class App {

    public static void main(String[] args) {
            Articulo[] articulos = {new Articulo("A1", 10.0), new Articulo("A2", 30.5), new Articulo("B1", 30.0), new
Articulo("B2", 100.0), new Articulo("c1", 66.5),};
        List<Articulo> listaArticulos = Arrays.asList(articulos);
        List<Articulo> filtrados;
        filtrados = listaArticulos
                .stream()
                .filter(a -> a.pvp>30)
                .sorted()
                .collect(Collectors.toList());
        System.out.println(filtrados);

    }
}
```

Ejercicio U6_B7_E5:

```
class Articulo {

    String codigo;
    double pvp;

    public Articulo(String codigo, double pvp) {
        this.codigo = codigo;
```

```
        this.pvp = pvp;
    }

    public String getCodigo() {
        return codigo;
    }

    public Double getPvp() {
        return pvp;
    }

}

public class App {

    public static void main(String[] args) {
            Articulo[] articulos = {new Articulo("A1", 10.0), new Articulo("A2", 30.5), new Articulo("B1", 30.0), new Articulo("B2", 100.0), new Articulo("c1", 66.5),};
        List<Articulo> listaArticulos = Arrays.asList(articulos);
        double total = listaArticulos
                .stream()
                .filter(a -> a.codigo.startsWith("B"))
                .mapToDouble(a -> a.getPvp())
                .sum();
        System.out.println(total);

    }
}
```

Nos fijamos bien en lo que devuelven map y mapToDouble()

`<R> Stream<R>`	**`map`**`(Function<? super T,? extends R> mapper)` Returns a stream consisting of the results of applying the given function to the elements of this stream.
`DoubleStream`	**`mapToDouble`**`(ToDoubleFunction<? super T> mapper)` Returns a `DoubleStream` consisting of the results of applying the given function to the elements of this stream.

map() devuelve un Stream y podemos comprobar que Stream no tiene ningún método sum(), en cambio, DoubleStream si tiene un método sum() que además devuelve un double

`double`	**`sum()`** Returns the sum of elements in this stream.

Ejercicio U6_B7_E6:

```
import java.util.Arrays;
import java.util.OptionalDouble;


class App{
    public static void main(String[] args) {
        String[] cadenas= {"a","ab","abc"};
        double media= Arrays.stream(cadenas).mapToInt(s->s.length()).average().getAsDouble();
        System.out.println(media);
    }
}
```

podemos mapear también con referencia a métodos, así queda más limpia la lectura de la tubería
mapToInt(String::length)