COMPARATOR

```
lista.sort((o1,o2)-> o1.getNacimiento() - o2.getNacimiento());
```

```
public static <T> Comparator<T> comparingInt(
                                    ToIntFunction<? super T> keyExtractor)
```

```
lista.sort(Comparator.comparingInt(Persona::getNacimiento));
```

```
class Persona {
    ...
    static Comparator<Persona> porEdad() {
        return Comparator.comparingInt(Persona::getNacimiento);
    }

}
```

```
lista.sort(Persona.porEdad());
```

```
lista.sort(Persona.porEdad().reversed());
```

Comparing

```
static Comparator<Persona> porApellido1() {
    return  Comparator.comparing(Persona::getApellido1);
}
```

```
public static <T, U> Comparator<T> comparing(
            Function<? super T, ? extends U> keyExtractor,
            Comparator<? super U> keyComparator)
```

Composición

```
default Comparator<T> thenComparing(Comparator<? super T> other)
```

```
lista.sort( Persona.porEdad()
                .thenComparing( Persona.porApellido1()));
```

nulls

```
public static <T> Comparator<T> nullsLast(
                                    Comparator<? super T> comparator)
```

```
lista.sort(  Comparator.nullsLast(
                    Persona.porEdad().thenComparing(
                    Persona.porApellido1()))
);
```

COMPARATOR