Creación de listas

```
List<String> arrayListDeStrings =  new ArrayList<String>();
arrayListDeStrings.add("Luciano");
arrayListDeStrings.add("Florencia");
arrayListDeStrings.add("Victoria");
```

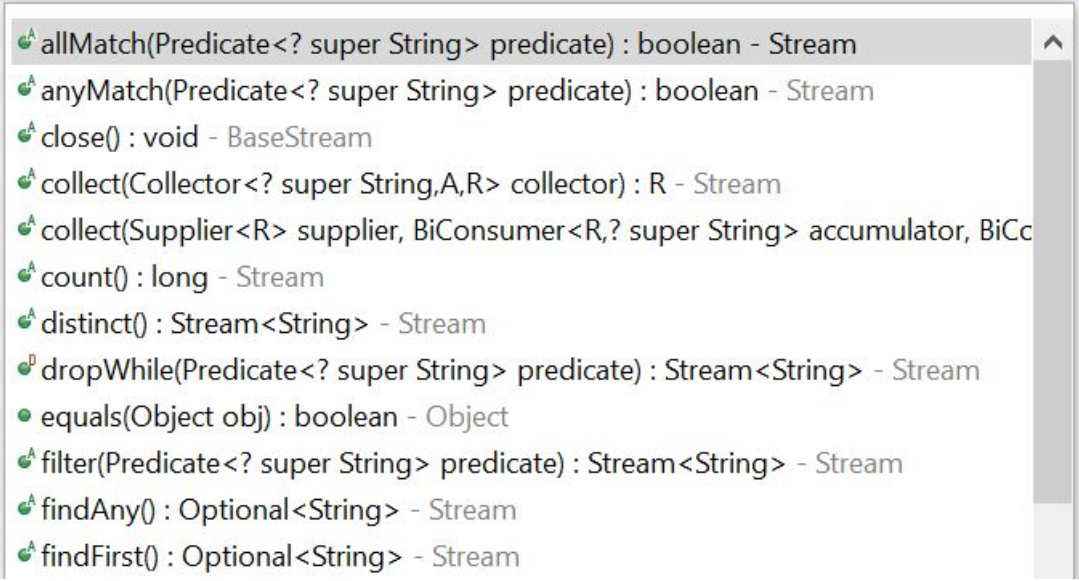
Camino corto con el mismo resultado

```
List<String> arrayListDeStrings =  Arrays.asList("Luciano","Florencia", "Victoria");
```

Streams

A sequence of elements supporting aggregate operations.

```
List<String> arrayListDeStrings =  Arrays.asList("Luciano","Florencia", "Victoria");
Stream<String> stream =  arrayListDeStrings.stream();
stream.
```



Stream foreach

```
List<String> arrayListDeStrings =  Arrays.asList("Luciano","Florencia", "Victoria");
Stream<String> stream =  arrayListDeStrings.stream();
stream.forEach(s -> System.out.println("Hola " + s));
```

Stream anyMatch, allMatch, noneMatch

¿Cómo hago en Java para evaluar si hay un nombre que empiece con L?

Stream anyMatch, allMatch, noneMatch

```
List<String> arrayListDeStrings =  Arrays.asList("Luciano","Florencia", "Victoria");
Stream<String> stream =  arrayListDeStrings.stream();
System.out.println(stream.anyMatch(s -> s.equals("Luciano")));
```

Stream filter

```
List<String> arrayListDeStrings =  Arrays.asList("Luciano","Florencia", "Victoria", "Luciana");
List<String> nombresQueEmpiezanConL = arrayListDeStrings.
        stream().
        filter(s -> s.startsWith("L")).
        collect(Collectors.toList());
nombresQueEmpiezanConL.stream().forEach(s -> System.out.println(s));
```

Stream filter

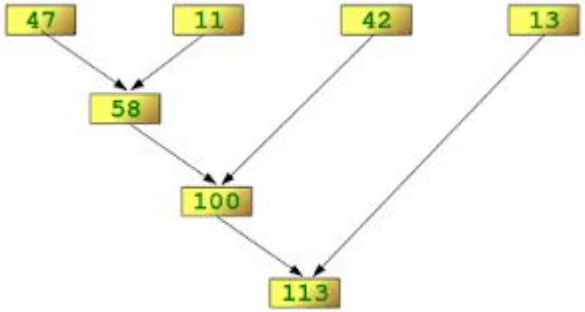
```
public List<MensajeColoreado> getMensajesDe(Asistente asistente) {

    List<MensajeColoreado> mensajes = this.getMensajes()
              .stream()
              .filter(m -> m.getAsistente().equals(asistente))
              .collect(Collectors.toList());

    return mensajes;

}
```

Stream reduce

```
List<Integer> arrayNumeros =  Arrays.asList(47,11,42,13);
int resultado = arrayNumeros.stream().reduce(0, (acumulado, nuevo) -> acumulado+nuevo);
```

Stream reduce

```
List<Integer> arrayNumeros =  Arrays.asList(47,11,42,13);
int resultado = arrayNumeros.stream().reduce(0, (acumulado, nuevo) -> acumulado+nuevo);
```



Stream reduce

```
List<Integer> arrayVacio =  Arrays.asList();
resultado = arrayVacio.stream().reduce(0, (acumulado, nuevo) -> acumulado+nuevo);
```

Stream reduce

```
List<ValueHolder> arrayVH = Arrays.asList(new ValueHolder(5), new ValueHolder(10), new ValueHolder(15));

ValueHolder resultado = arrayVH.stream().reduce(new ValueHolder(0),
        (acumulado, nuevo) -> new ValueHolder(acumulado.getValue() + nuevo.getValue()));
```

mapTo

```
int r = arrayVH.stream().mapToInt(p->p.getValue()).reduce(0,(acumulado, nuevo) -> acumulado+nuevo);
```

Anidamiento

```
ArrayList<Producto> lista= new ArrayList<Producto>();

lista.add(new Producto("Milanesa",80));
lista.add(new Producto("Puré",50));
lista.add(new Producto("Agua",70));
lista.add(new Producto("Fideo",95));

double resultado=lista.stream()
.mapToDouble(producto->producto.getMonto()*1.21)
.filter(producto->producto<100)
.sum();

System.out.println(resultado);
```

Anidamiento

Pueden leer la interfaz Stream

```
interface FuncInterface
{
    // An abstract function
    void abstractFun(int x);

    // A non-abstract (or default) function
    default void normalFun()
    {
       System.out.println("Hello");
    }
}

class Test
{
    public static void main(String args[])
    {
        // lambda expression to implement above
        // functional interface. This interface
        // by default implements abstractFun()
        FuncInterface fobj = (int x)->System.out.println(2*x);

        // This calls above lambda expression and prints 10.
        fobj.abstractFun(5);
    }
}
```