**1 Create a list collection and assign a value**

The most primitive way:

List<String> stringList = new ArrayList<>();  
stringList.add("jack");  
stringList.add("pony");  
stringList.add("ekko");

Simple Retrofit:

List<String> stringList2 = new ArrayList<String>(4) {{  
 add("jack");  
 add("pony");  
 add("ekko");  
}};

The ultimate makeover:

List<String> stringList3 = ImmutableList.of("jack", "pony", "ekko");

# 2 Remove empty values from a list

General practice:

List<String> nameList = new ArrayList<>();  
List<String> noNullList = new ArrayList<>();  
nameList.add("jack");  
nameList.add("pony");  
nameList.add("ekko");  
nameList.add(null);  
for (String o : stringList) {  
 if (o != null) {  
 noNullList.add(o);  
 }  
}

Simplified writing using [lambda](https://medium.com/javarevisited/7-best-java-tutorials-and-books-to-learn-lambda-expression-and-stream-api-and-other-features-3083e6038e14?source=---------14------------------):

List<String> noNullListFun = nameList  
 .stream()  
 .filter(Objects::nonNull)  
 .collect(Collectors.toList());

# 3 Sum the values in a list

Original practice:

List<BigDecimal> numList = new ArrayList<BigDecimal>(10) {{  
 add(BigDecimal.valueOf(111L));  
 add(BigDecimal.valueOf(8888.22));  
 add(BigDecimal.valueOf(333.22));  
 add(BigDecimal.valueOf(857857.22));  
 add(BigDecimal.valueOf(5331.22));  
}};  
BigDecimal total = BigDecimal.ZERO;  
for (BigDecimal num : numList) {  
 total = total.add(num);  
}  
System.out.println(total);

To simplify:

List<BigDecimal> numListSimple = ImmutableList.of(BigDecimal.valueOf(111L)  
 , BigDecimal.valueOf(8888.22), BigDecimal.valueOf(333.22)  
 , BigDecimal.valueOf(857857.22), BigDecimal.valueOf(5331.22));  
BigDecimal totalNum = BigDecimal.valueOf(numListSimple.stream().mapToDouble(BigDecimal::doubleValue).sum());

# 4 Determine whether the value is empty

Original spelling:

public static String getUserName() {  
 return null;  
}  
String username = getUserName();  
String res;  
if (username != null) {  
 res = username;  
} else {  
 res = "Tom";  
}

simplify:

String userName = Optional.ofNullable(username).orElse("Tom");  
System.out.println(userName);

# 5 Get matching values from multiple sets

Original method:

package com.github.springtools.test;import com.google.common.collect.ImmutableList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;public class TestFlatMap {  
 public static void main(String[] args) {  
 List<String> nameListA = ImmutableList.of("Cat01", "Dog02");  
 List<String> nameListB = ImmutableList.of("Mouse01", "Dog22");  
 List<String> nameListC = ImmutableList.of("Pig02", "Dog00");  
 Set<String> nameSet = new HashSet<>();  
 nameListA.forEach(n -> {  
 if (n.startsWith("Dog")) {  
 nameSet.add(n);  
 }  
 });  
 nameListB.forEach(n -> {  
 if (n.startsWith("Dog")) {  
 nameSet.add(n);  
 }  
 });  
 nameListC.forEach(n -> {  
 if (n.startsWith("Dog")) {  
 nameSet.add(n);  
 }  
 });  
 System.out.println(nameSet.toString());  
 }  
}

Modified code:

import com.google.common.collect.ImmutableList;import java.util.List;  
import java.util.Set;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;public class TestFlatMap2 {  
 public static void main(String[] args) {  
  
 List<String> nameListA = ImmutableList.of("Cat01", "Dog02");  
 List<String> nameListB = ImmutableList.of("Mouse01", "Dog22");  
 List<String> nameListC = ImmutableList.of("Pig02", "Dog00");  
 Set<String> nameSet = Stream.of(nameListA, nameListB, nameListC)  
 .flatMap(list -> list.stream().filter(name -> name.startsWith("Dog")))  
 .collect(Collectors.toSet());  
 System.out.println(nameSet.toString());  
 }  
}