public Stream map(Function<? super T,? extends R> mapper) :

It is a predefined method of Stream interface.

It takes Function (Predefined functional interafce) as a parameter.

It performs intermediate operation and consumes single element from input Stream and produces single element to output Stream. (1:1 transformation)

Here mapper function is functional interface which takes one input and provides one output.

```java
package com.ravi.map;

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

public class MapDemo1
{
    public static void main(String[] args)
    {
       List<Integer> listOfNumbers = Arrays.asList(1,2,3,4,5,6,7,8,9,10);
       //add a constant value 10 to all the numbers

       List<Integer> list = listOfNumbers.stream().map(num -> num+10).toList();
       System.out.println(list);

       System.out.println("................................");

      List<Integer> immutableList = List.of(1,2,3,4,5,6,7,8,9,10,2,3,4,6,8);

      //Fetch all the unique even numbers and find the cube of those numbers

      immutableList.stream()
                    .distinct()
                       .filter(num -> num%2==0)
                           .map(n -> n*n*n)
                              .forEach(System.out::println);



    }
}
```

default List<T> toList() :
----

* It is a predefined method of Stream interface available from JDK 16v.
* It is used to convert the Stream into List (Stream to Collection)

```java
  package com.ravi.map;

  import java.util.ArrayList;
  import java.util.List;
  import java.util.stream.Collectors;

  record Employee(Integer id, String name, Double salary)
  {

  }

  public class MapDemo2
  {
      public static void main(String[] args)
      {
          ArrayList<Employee> listOfEmp = new ArrayList<>();
          listOfEmp.add(new Employee(1, "Scott", 800D));
          listOfEmp.add(new Employee(2, "Smith", 1200D));
          listOfEmp.add(new Employee(3, "Alen", 1500D));
          listOfEmp.add(new Employee(4, "Martin", 1800D));
          listOfEmp.add(new Employee(5, "John", 2000D));

          System.out.println("Original Employee Data with base Salary");
          listOfEmp.forEach(System.out::println);

          //add 500D in the salary for all the Employees
          List<String> collect = listOfEmp.stream().map(emp -> emp + " updated Salary
  "+(emp.salary()+500)).toList();

          System.out.println("Employee Data after Salary Increment");
          collect.forEach(System.out::println);


      }
  }
```

```java
package com.ravi.map;

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

record Player(Integer id, String name)
{

}

public class MapDemo3
{
    public static void main(String args[])
    {
        //Get the name of the Player in upper-case from Player Object where duplicate
        //should not be there

        createPlayerList().stream()
                 .map(player -> player.name().toUpperCase())
                     .distinct()
                        .forEach(System.out::println);
    }


    public static List<Player> createPlayerList()
    {
        List<Player> al = new ArrayList<>();
        al.add(new Player(18, "Virat"));
        al.add(new Player(45, "Rohit"));
        al.add(new Player(7, "Dhoni"));
        al.add(new Player(18, "Virat"));
        al.add(new Player(90, "Bumrah"));
        al.add(new Player(67, "Hardik"));

        return al;
    }
}
```

```java
package com.ravi.map;

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

//Retrieve first character of all the given name
public class MapDemo4
{
    public static void main(String[] args)
    {
      List<String> listOfName = Arrays.asList("Jaya","Arnav","Virat","Aryan");

      List<Character> list = listOfName
         .stream()
             .map(name -> name.charAt(0))
                 .toList();

      System.out.println(list);

    }

}
```

public Stream flatMap(Function<? super T,? extends Stream<? extends R>> mapper)

It is a predefined method of Stream interface.

The map() method produces one output value for each input value in the stream So if there are n elements in the stream, map() operation will produce a stream of n output elements.

flatMap() is two step process i.e. map() + Flattening. It helps in converting Collection<Collection<T>> into Collection<T> [to make flat i.e converting Collections of collection into single collection or merging of all the collections into single Collection]

It is mainly used to deal with nested structure.

```java
package com.ravi.flatmap;

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

public class FlatMapDemo1
{
    public static void main(String[] args)
    {
        List<String> indianPlayer = Arrays.asList("Rohit", "Virat", "Gill", "Bumrah");
        List<String> engPlayer = Arrays.asList("Stoke", "Root", "Brook", "Butler");

        List<List<String>> ipl = Arrays.asList(indianPlayer, engPlayer);
        System.out.println(ipl);

        System.out.println(".......................");

        List<String> flatteningColl = ipl.stream()
             .flatMap(team -> team.stream())
                   .collect(Collectors.toList());

        System.out.println(flatteningColl);

    }

}
```

```java
 package com.ravi.flatmap;

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

 public class FlatMapDemo2 {

     public static void main(String[] args)
     {
         List<String> list1 = Arrays.asList("A","B","C");
         List<String> list2 = Arrays.asList("D","E","F");
         List<String> list3 = Arrays.asList("G","H","I");

         List<List<String>> nestedColl = Arrays.asList(list1, list2, list3);
         System.out.println(nestedColl);

         System.out.println("....................");


         Set<String> flattening = nestedColl.stream()
                     .flatMap(list -> list.stream())
                                         .collect(Collectors.toSet());
         System.out.println(flattening);

     }

 }
```

```java
package com.ravi.flatmap;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class FlatMapDemo3
{
    //Fetching first character using flatMap()

    public static void main(String[] args)
    {
        List<String> listOfNames = Arrays.asList("Jaya","Aryan","Virat","Aakash");

        listOfNames
            .stream()
                .flatMap(str -> Stream.of(str.charAt(0)))
                           .forEach(System.out::print);

    }

}
```

```java
package com.ravi.flatmap;

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

record Product(Integer id, List<String> listOfProducts)
{

}

public class FlatMapDemo4
{
    public static void main(String[] args)
    {
        List<Product> listOfProduct = Arrays.asList(
                new Product(1, Arrays.asList("Camera", "Mobile", "Laptop")),
                new Product(2, Arrays.asList("Bat", "Ball", "Wicket")),
                new Product(3, Arrays.asList("Chair", "Table", "Lamp")),
                new Product(4, Arrays.asList("Cycle", "Bike", "Car"))

                    );


        System.out.println(listOfProduct);
        System.out.println("........................");

        List<String> flattening = listOfProduct
                  .stream()
                       .flatMap(product -> product.listOfProducts().stream())
                           .toList();
        System.out.println(flattening);

    }

}
```