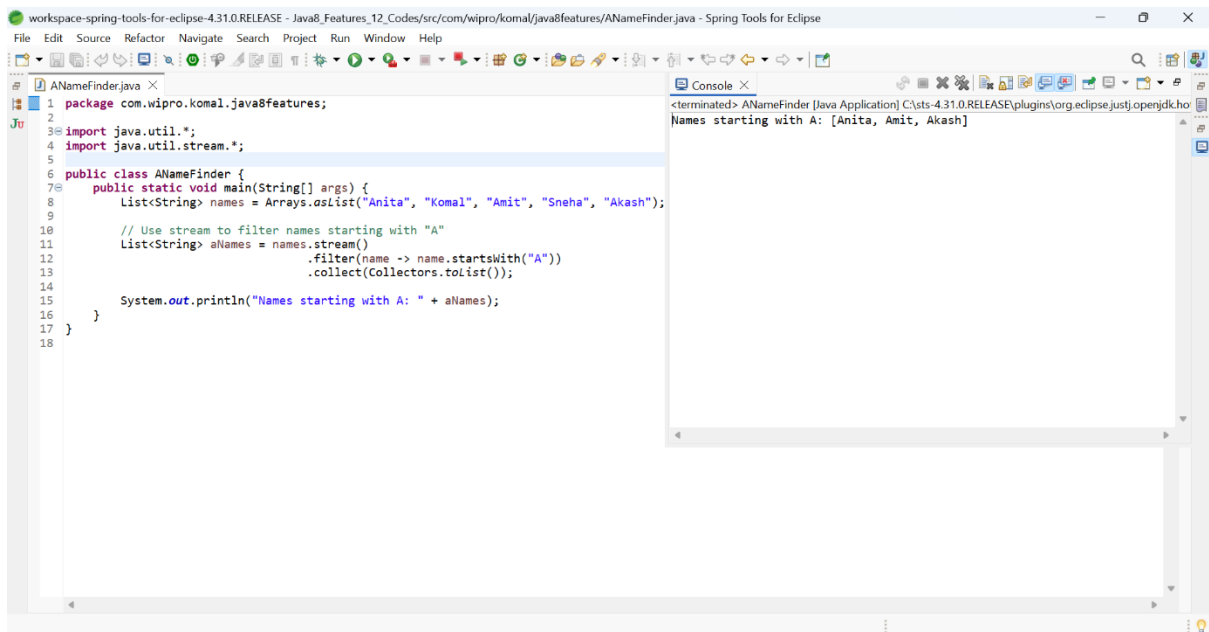


Name:- Komal Sudhakar Baviskar

## #Java\_8\_features codes Output

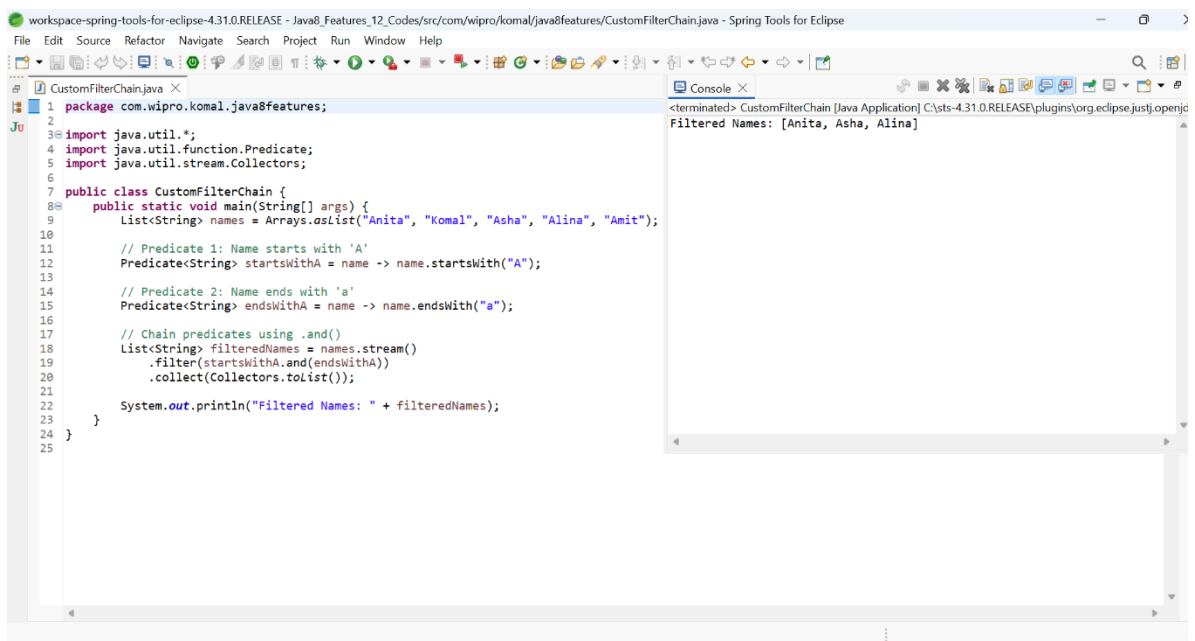
1)



```
workspace-spring-tools-for-eclipse-4.31.0.RELEASE - Java8_Features_12_Codes/src/com/wipro/komal/java8features/ANameFinder.java - Spring Tools for Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
ANameFinder.java
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 import java.util.stream.*;
5
6 public class ANameFinder {
7     public static void main(String[] args) {
8         List<String> names = Arrays.asList("Anita", "Komal", "Amit", "Sneha", "Akash");
9
10        // Use stream to filter names starting with "A"
11        List<String> aNames = names.stream()
12                                .filter(name -> name.startsWith("A"))
13                                .collect(Collectors.toList());
14
15        System.out.println("Names starting with A: " + aNames);
16    }
17 }
18

Console
<terminated> ANameFinder [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full\jre\bin\java.exe
Names starting with A: [Anita, Amit, Akash]
```

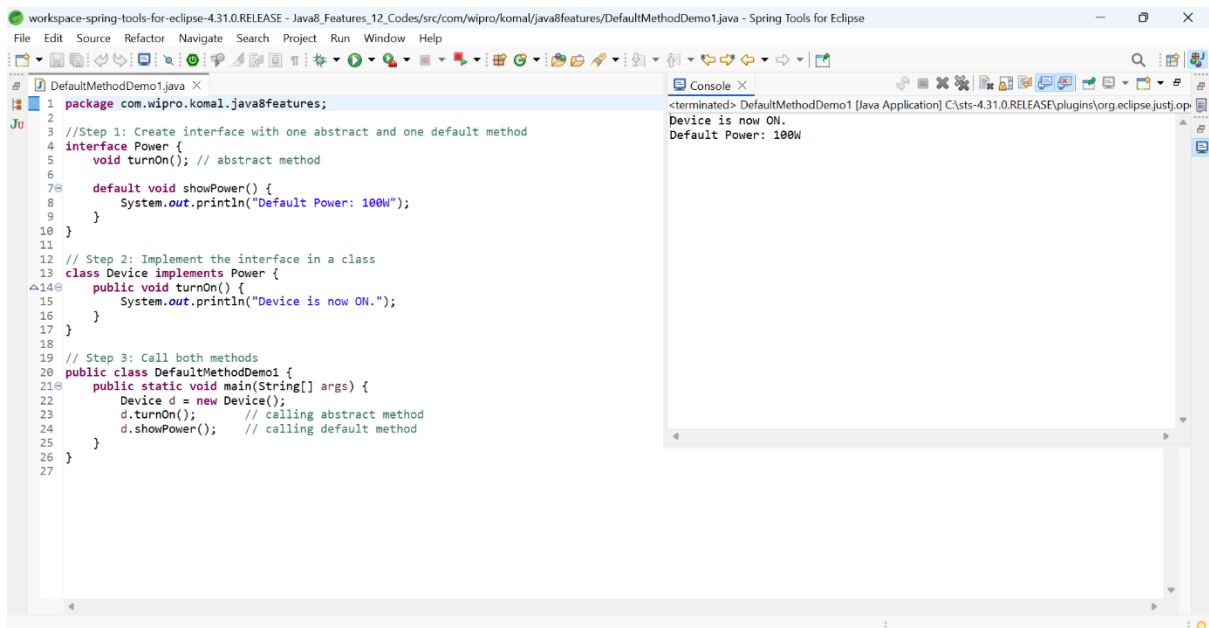
2)



```
workspace-spring-tools-for-eclipse-4.31.0.RELEASE - Java8_Features_12_Codes/src/com/wipro/komal/java8features/CustomFilterChain.java - Spring Tools for Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
CustomFilterChain.java
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 import java.util.function.Predicate;
5 import java.util.stream.Collectors;
6
7 public class CustomFilterChain {
8     public static void main(String[] args) {
9         List<String> names = Arrays.asList("Anita", "Komal", "Asha", "Alina", "Amit");
10
11        // Predicate 1: Name starts with 'A'
12        Predicate<String> startsWithA = name -> name.startsWith("A");
13
14        // Predicate 2: Name ends with 'a'
15        Predicate<String> endsWithA = name -> name.endsWith("a");
16
17        // Chain predicates using .and()
18        List<String> filteredNames = names.stream()
19                                .filter(startsWithA.and(endsWithA))
20                                .collect(Collectors.toList());
21
22        System.out.println("Filtered Names: " + filteredNames);
23    }
24 }
25

Console
<terminated> CustomFilterChain [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full\jre\bin\java.exe
Filtered Names: [Anita, Asha, Alina]
```

3)



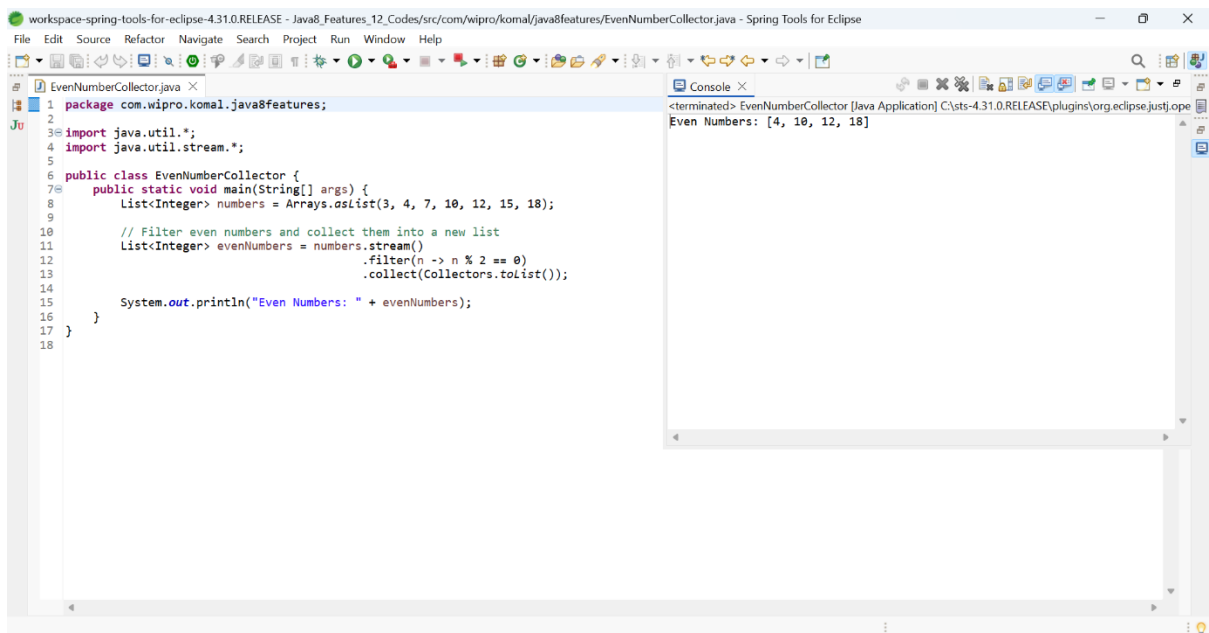
The screenshot shows the Eclipse IDE with the file `DefaultMethodDemo1.java` open. The code is as follows:

```
1 package com.wipro.komal.java8features;
2
3 //Step 1: Create interface with one abstract and one default method
4 interface Power {
5     void turnOn(); // abstract method
6
7     default void showPower() {
8         System.out.println("Default Power: 100W");
9     }
10 }
11
12 // Step 2: Implement the interface in a class
13 class Device implements Power {
14     public void turnOn() {
15         System.out.println("Device is now ON.");
16     }
17 }
18
19 // Step 3: Call both methods
20 public class DefaultMethodDemo1 {
21     public static void main(String[] args) {
22         Device d = new Device();
23         d.turnOn(); // calling abstract method
24         d.showPower(); // calling default method
25     }
26 }
27
```

The console output on the right shows the following messages:

```
<terminated> DefaultMethodDemo1 [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj...
Device is now ON.
Default Power: 100W
```

4)



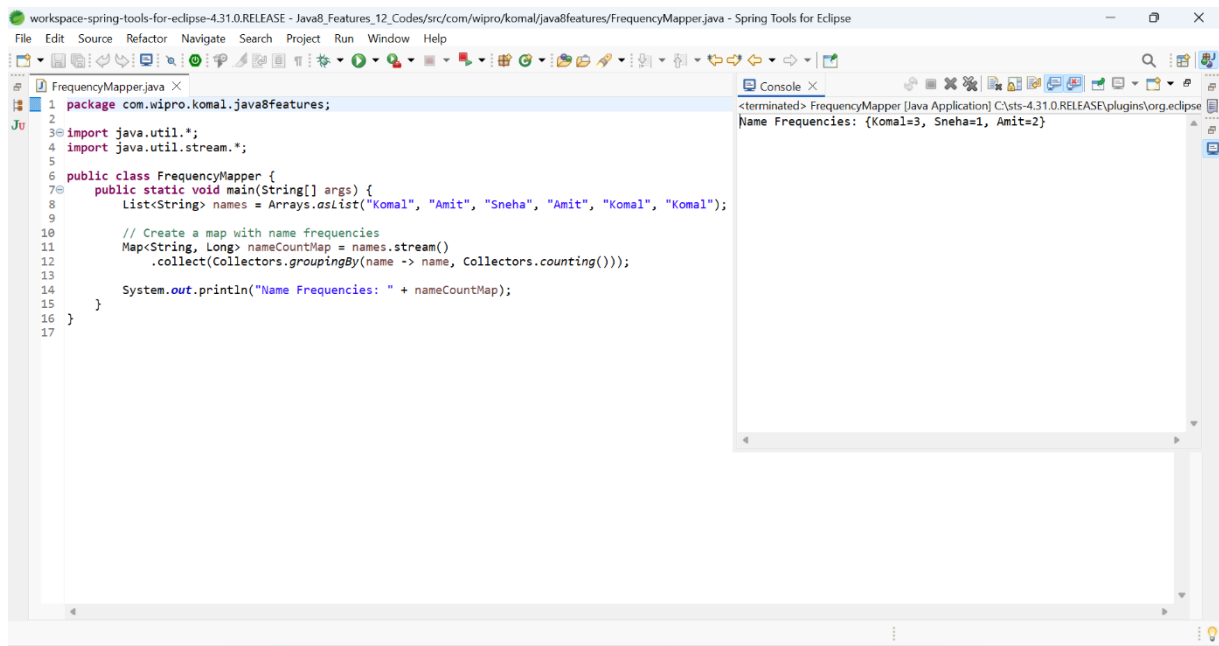
The screenshot shows the Eclipse IDE with the file `EvenNumberCollector.java` open. The code is as follows:

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 import java.util.stream.*;
5
6 public class EvenNumberCollector {
7     public static void main(String[] args) {
8         List<Integer> numbers = Arrays.asList(3, 4, 7, 10, 12, 15, 18);
9
10         // Filter even numbers and collect them into a new list
11         List<Integer> evenNumbers = numbers.stream()
12             .filter(n -> n % 2 == 0)
13             .collect(Collectors.toList());
14
15         System.out.println("Even Numbers: " + evenNumbers);
16     }
17 }
18
```

The console output on the right shows the following message:

```
<terminated> EvenNumberCollector [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj...
Even Numbers: [4, 10, 12, 18]
```

5)



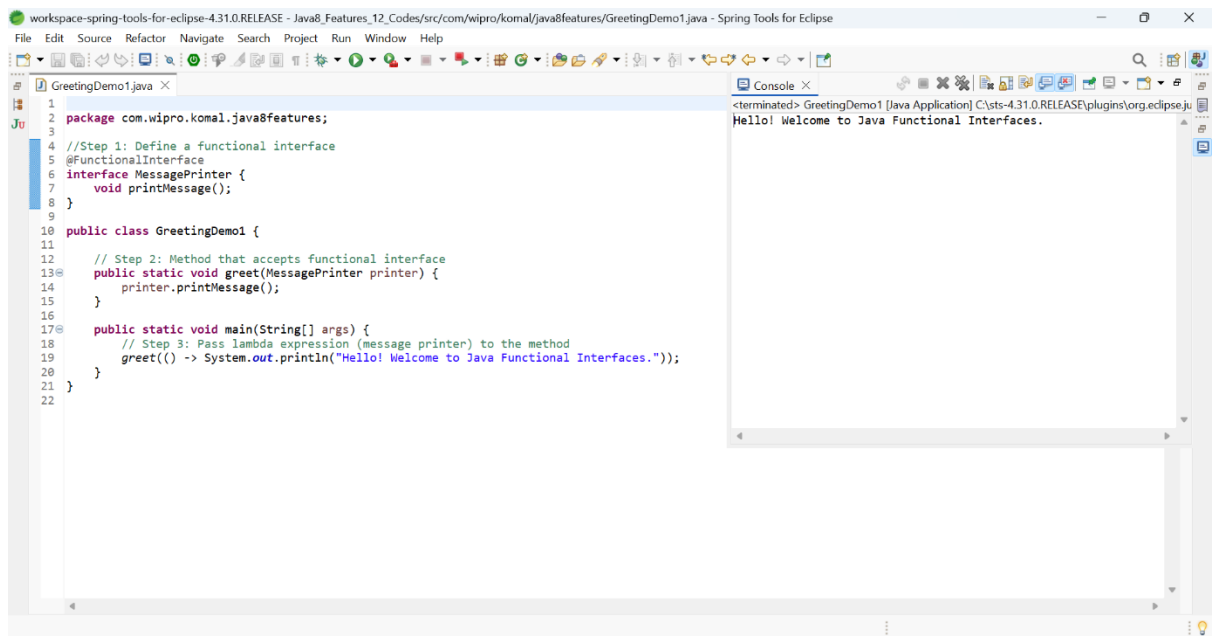
The screenshot shows the Eclipse IDE with a Java project. The editor displays the file `FrequencyMapper.java` with the following code:

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 import java.util.stream.*;
5
6 public class FrequencyMapper {
7     public static void main(String[] args) {
8         List<String> names = Arrays.asList("Komal", "Amit", "Sneha", "Amit", "Komal", "Komal");
9
10        // Create a map with name frequencies
11        Map<String, Long> nameCountMap = names.stream()
12            .collect(Collectors.groupingBy(name -> name, Collectors.counting()));
13
14        System.out.println("Name Frequencies: " + nameCountMap);
15    }
16 }
17
```

The console on the right shows the output of the program:

```
<terminated> FrequencyMapper [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse
Name Frequencies: {Komal=3, Sneha=1, Amit=2}
```

6)



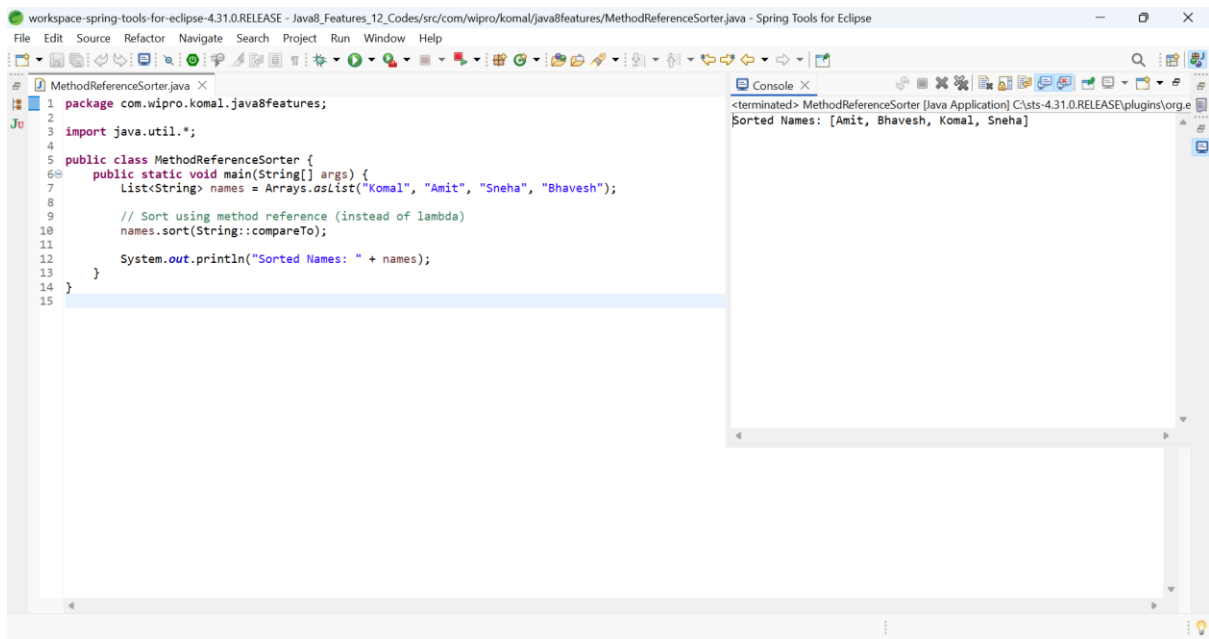
The screenshot shows the Eclipse IDE with a Java project. The editor displays the file `GreetingDemo1.java` with the following code:

```
1 package com.wipro.komal.java8features;
2
3 //Step 1: Define a functional interface
4 @FunctionalInterface
5 interface MessagePrinter {
6     void printMessage();
7 }
8
9 public class GreetingDemo1 {
10
11    // Step 2: Method that accepts functional interface
12    public static void greet(MessagePrinter printer) {
13        printer.printMessage();
14    }
15
16    public static void main(String[] args) {
17        // Step 3: Pass lambda expression (message printer) to the method
18        greet(() -> System.out.println("Hello! Welcome to Java Functional Interfaces."));
19    }
20 }
21
22
```

The console on the right shows the output of the program:

```
<terminated> GreetingDemo1 [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.ju
Hello! Welcome to Java Functional Interfaces.
```

7)

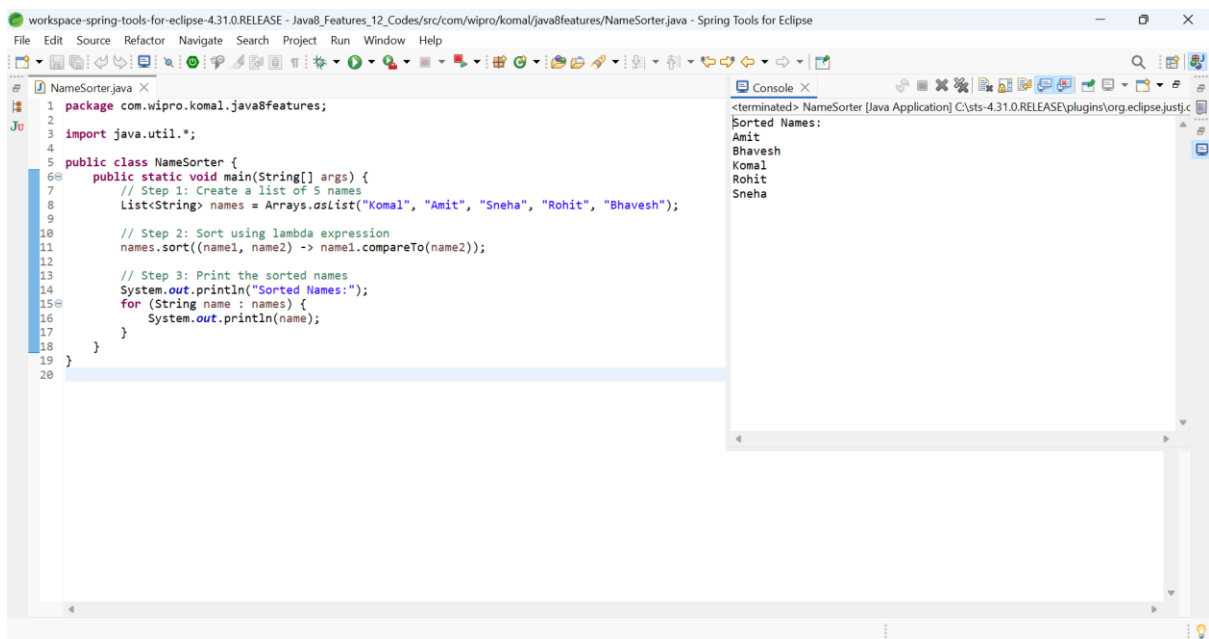


The screenshot shows the Eclipse IDE with the file `MethodReferenceSorter.java` open. The code defines a class `MethodReferenceSorter` with a `main` method that sorts an array of names using `String::compareTo`. The console output shows the sorted names: `[Amit, Bhavesh, Komal, Sneha]`.

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4
5 public class MethodReferenceSorter {
6     public static void main(String[] args) {
7         List<String> names = Arrays.asList("Komal", "Amit", "Sneha", "Bhavesh");
8
9         // Sort using method reference (instead of lambda)
10        names.sort(String::compareTo);
11
12        System.out.println("Sorted Names: " + names);
13    }
14 }
15
```

Console Output:  
<terminated> MethodReferenceSorter [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.e  
Sorted Names: [Amit, Bhavesh, Komal, Sneha]

8)

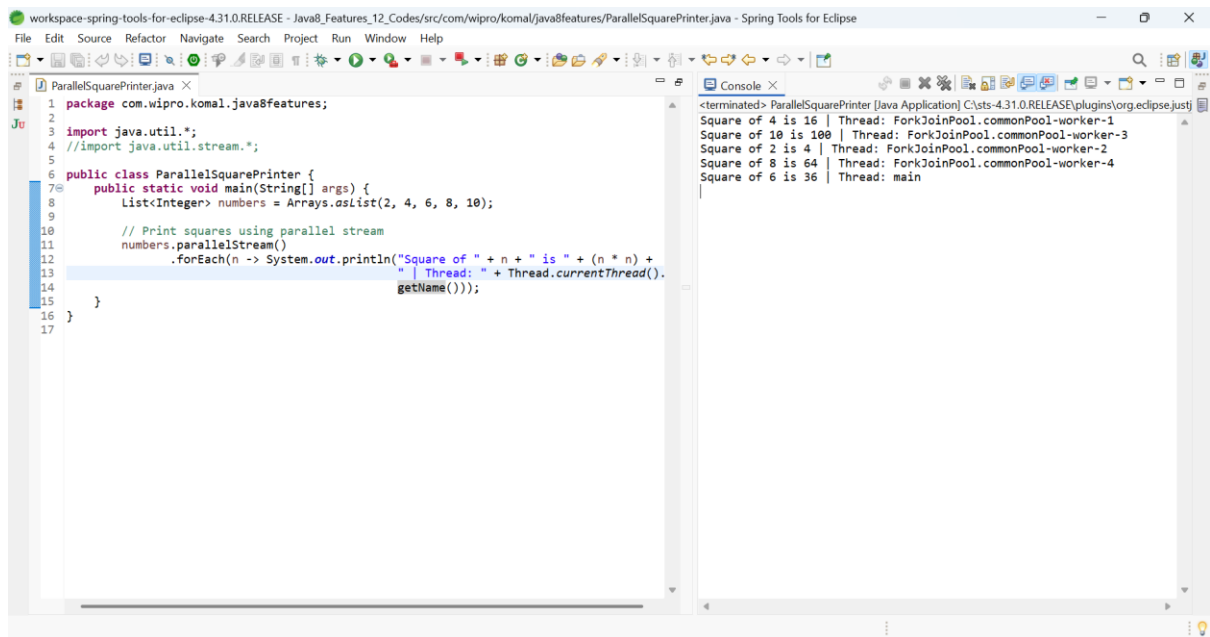


The screenshot shows the Eclipse IDE with the file `NameSorter.java` open. The code defines a class `NameSorter` with a `main` method that sorts an array of names using a lambda expression and then prints each name. The console output shows the sorted names: `Amit, Bhavesh, Komal, Rohit, Sneha`.

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4
5 public class NameSorter {
6     public static void main(String[] args) {
7         // Step 1: Create a list of 5 names
8         List<String> names = Arrays.asList("Komal", "Amit", "Sneha", "Rohit", "Bhavesh");
9
10        // Step 2: Sort using lambda expression
11        names.sort((name1, name2) -> name1.compareTo(name2));
12
13        // Step 3: Print the sorted names
14        System.out.println("Sorted Names:");
15        for (String name : names) {
16            System.out.println(name);
17        }
18    }
19 }
20
```

Console Output:  
<terminated> NameSorter [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj.c  
Sorted Names:  
Amit  
Bhavesh  
Komal  
Rohit  
Sneha

9)



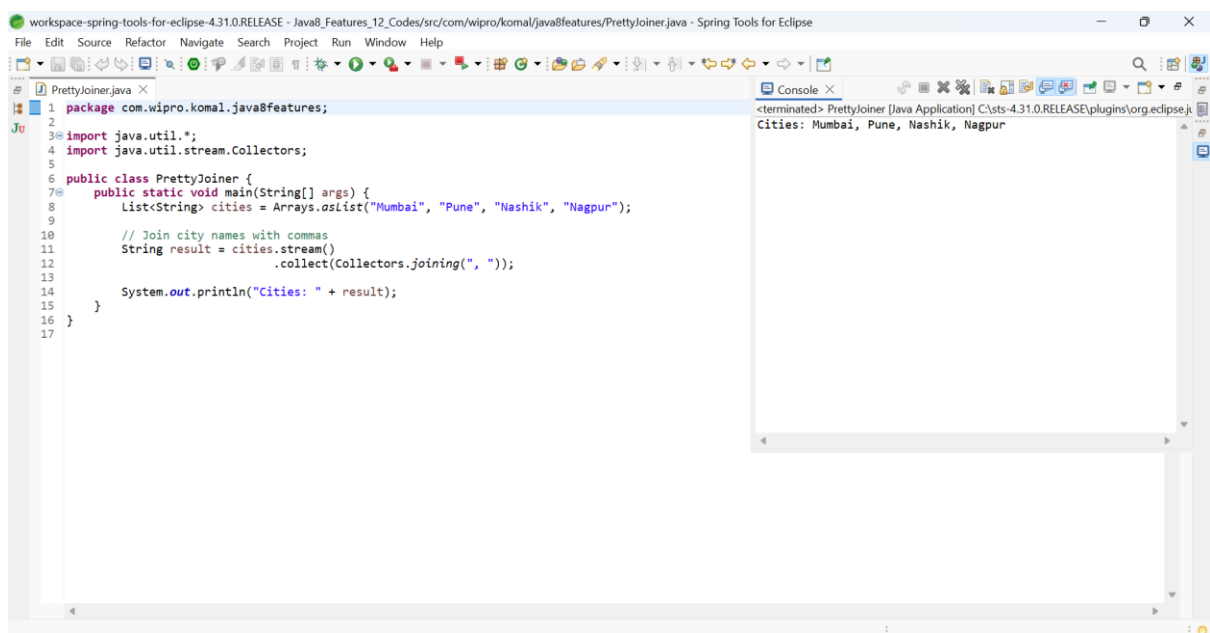
The screenshot shows the Eclipse IDE with a Java project named 'workspace-spring-tools-for-eclipse-4.31.0.RELEASE'. The editor displays the file 'ParallelSquarePrinter.java' with the following code:

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 //import java.util.stream.*;
5
6 public class ParallelSquarePrinter {
7     public static void main(String[] args) {
8         List<Integer> numbers = Arrays.asList(2, 4, 6, 8, 10);
9
10        // Print squares using parallel stream
11        numbers.parallelStream()
12            .forEach(n -> System.out.println("Square of " + n + " is " + (n * n) +
13                                           " | Thread: " + Thread.currentThread().
14                                           getName()));
15    }
16 }
17
```

The console output shows the execution results:

```
<terminated> ParallelSquarePrinter [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.jst.j2ee
Square of 4 is 16 | Thread: ForkJoinPool.commonPool-worker-1
Square of 10 is 100 | Thread: ForkJoinPool.commonPool-worker-3
Square of 2 is 4 | Thread: ForkJoinPool.commonPool-worker-2
Square of 8 is 64 | Thread: ForkJoinPool.commonPool-worker-4
Square of 6 is 36 | Thread: main
```

10)



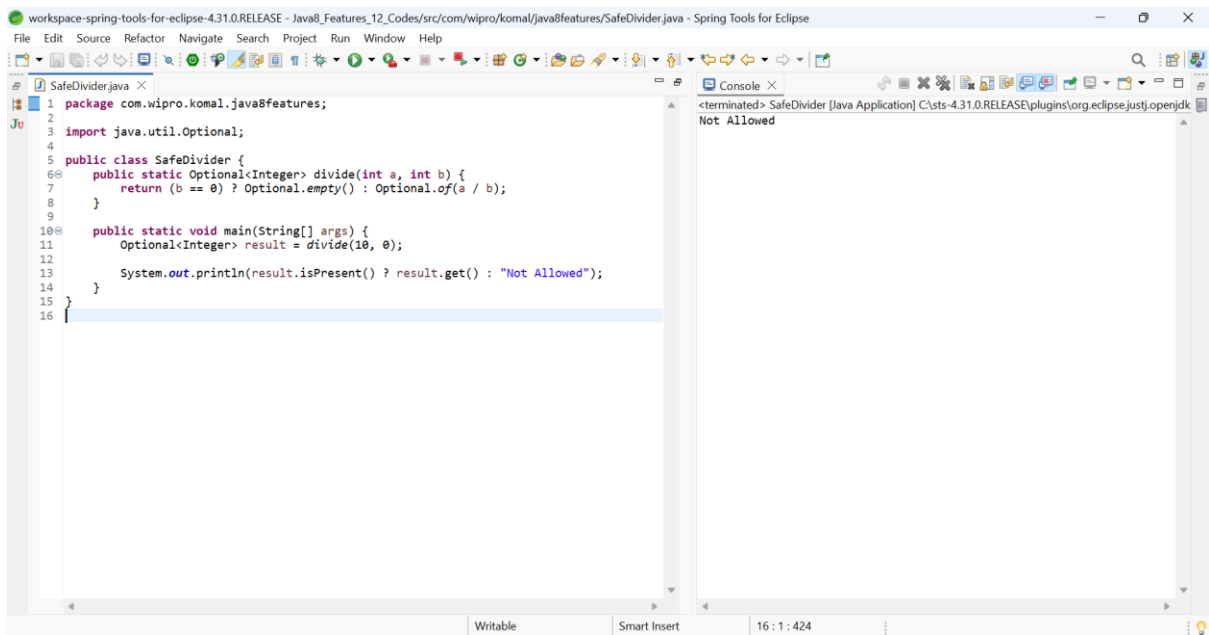
The screenshot shows the Eclipse IDE with a Java project named 'workspace-spring-tools-for-eclipse-4.31.0.RELEASE'. The editor displays the file 'PrettyJoiner.java' with the following code:

```
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 import java.util.stream.Collectors;
5
6 public class PrettyJoiner {
7     public static void main(String[] args) {
8         List<String> cities = Arrays.asList("Mumbai", "Pune", "Nashik", "Nagpur");
9
10        // Join city names with commas
11        String result = cities.stream()
12            .collect(Collectors.joining(", "));
13
14        System.out.println("Cities: " + result);
15    }
16 }
17
```

The console output shows the execution result:

```
<terminated> PrettyJoiner [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.jst.j2ee
Cities: Mumbai, Pune, Nashik, Nagpur
```

11)



The screenshot shows the Eclipse IDE with the file `SafeDivider.java` open. The code defines a `SafeDivider` class with a `divide` method that returns an `Optional<Integer>` and a `main` method that calls `divide(10, 0)` and prints the result. The console output shows the program terminated with the message "Not Allowed".

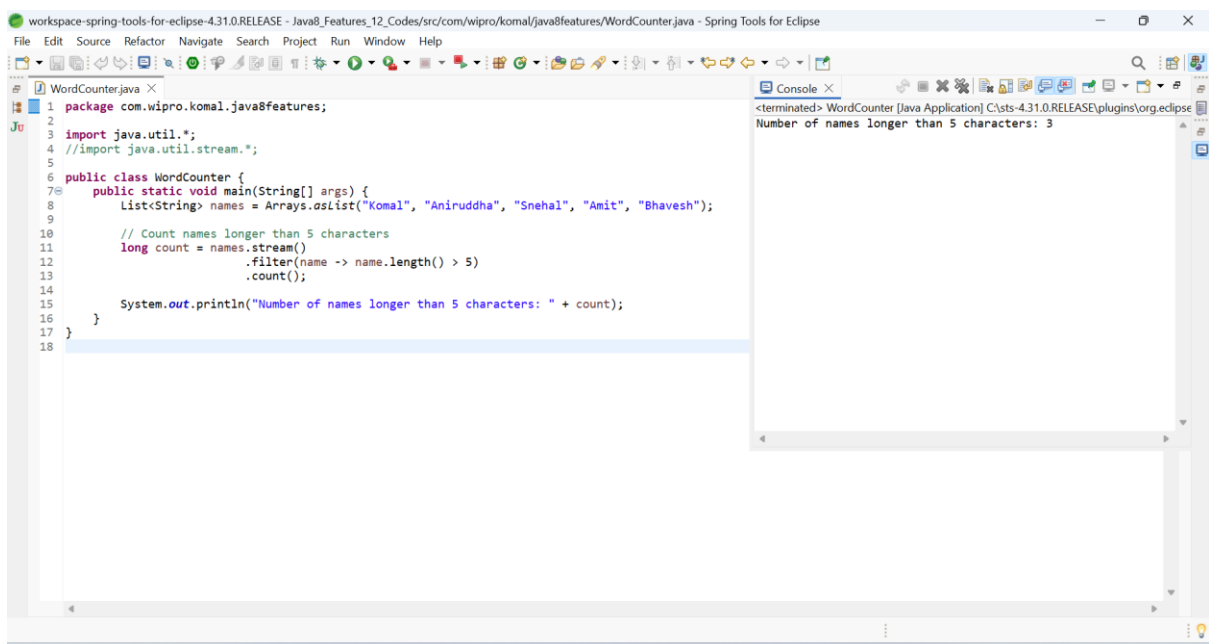
```
workspace-spring-tools-for-eclipse-4.31.0.RELEASE - Java8_Features_12_Codes/src/com/wipro/komal/java8features/SafeDivider.java - Spring Tools for Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

SafeDivider.java
1 package com.wipro.komal.java8features;
2
3 import java.util.Optional;
4
5 public class SafeDivider {
6     public static Optional<Integer> divide(int a, int b) {
7         return (b == 0) ? Optional.empty() : Optional.of(a / b);
8     }
9
10    public static void main(String[] args) {
11        Optional<Integer> result = divide(10, 0);
12        System.out.println(result.isPresent() ? result.get() : "Not Allowed");
13    }
14 }
15
16

Console
<terminated> SafeDivider [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse.justj.openjdk
Not Allowed

Writable Smart Insert 16 : 1 : 424
```

12)



The screenshot shows the Eclipse IDE with the file `WordCounter.java` open. The code defines a `WordCounter` class with a `main` method that counts the number of names longer than 5 characters in a list. The console output shows the program terminated with the message "Number of names longer than 5 characters: 3".

```
workspace-spring-tools-for-eclipse-4.31.0.RELEASE - Java8_Features_12_Codes/src/com/wipro/komal/java8features/WordCounter.java - Spring Tools for Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

WordCounter.java
1 package com.wipro.komal.java8features;
2
3 import java.util.*;
4 //import java.util.stream.*;
5
6 public class WordCounter {
7     public static void main(String[] args) {
8         List<String> names = Arrays.asList("Komal", "Aniruddha", "Snehal", "Amit", "Bhavesh");
9
10        // Count names longer than 5 characters
11        long count = names.stream()
12            .filter(name -> name.length() > 5)
13            .count();
14
15        System.out.println("Number of names longer than 5 characters: " + count);
16    }
17 }
18

Console
<terminated> WordCounter [Java Application] C:\sts-4.31.0.RELEASE\plugins\org.eclipse
Number of names longer than 5 characters: 3
```