

Lesson 05 Demo 05

Implementing HashMap and Hashtable

Objective: To demonstrate how to use HashMap, LinkedHashMap, and Hashtable in Java for efficient data handling

Tools Required: Eclipse IDE

Prerequisites: None

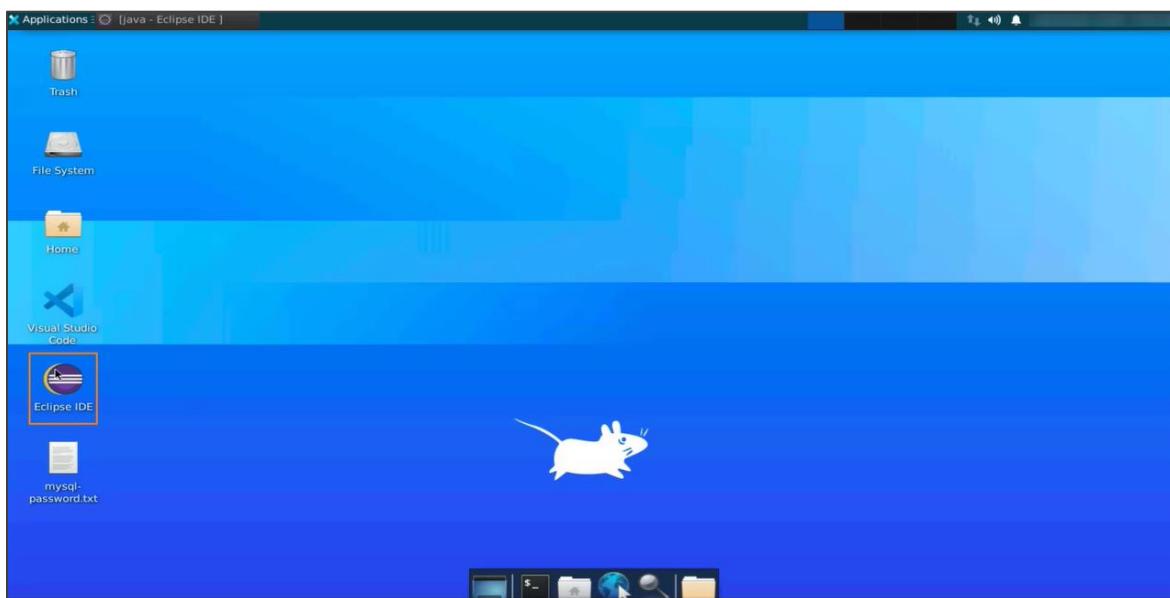
Steps to be followed:

1. Create a new project
2. Use a HashMap
3. Execute the code with example data
4. Sort the data based on keys
5. Iterate the data structure with example data
6. Use the remove() method and execute the code
7. Implement iteration to obtain all the keys from the map
8. Execute the entrySet() method and iterate through the code

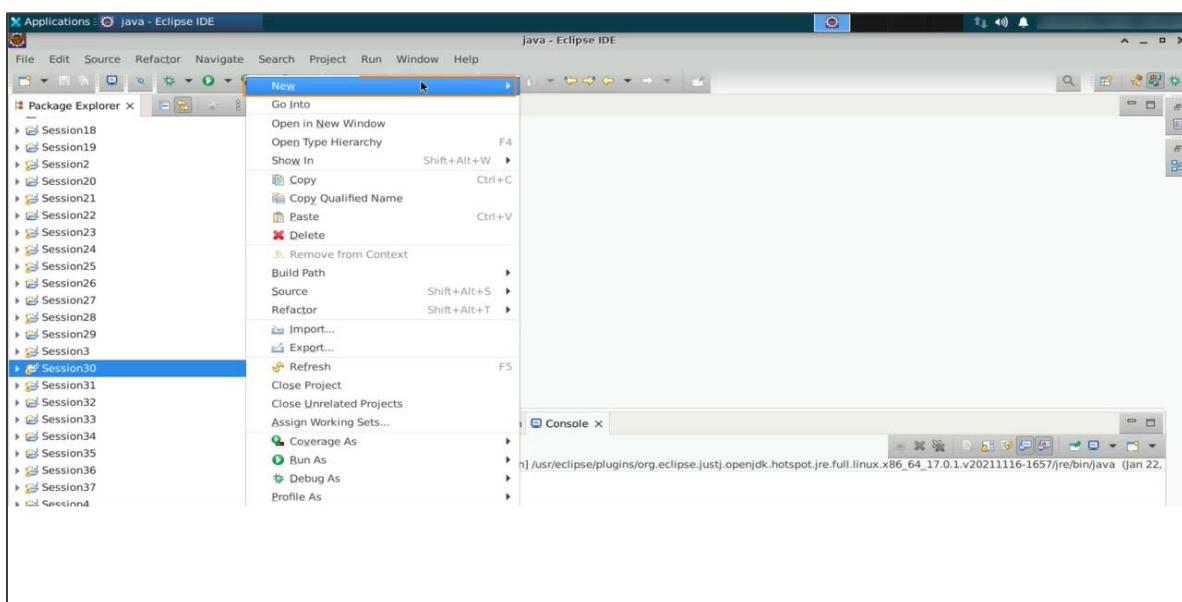
Step 1: Create a new project

1.1 In the Collections framework, you will explore the usage of a data structure called Map, which stores data as key-value pairs. It is an essential data structure.

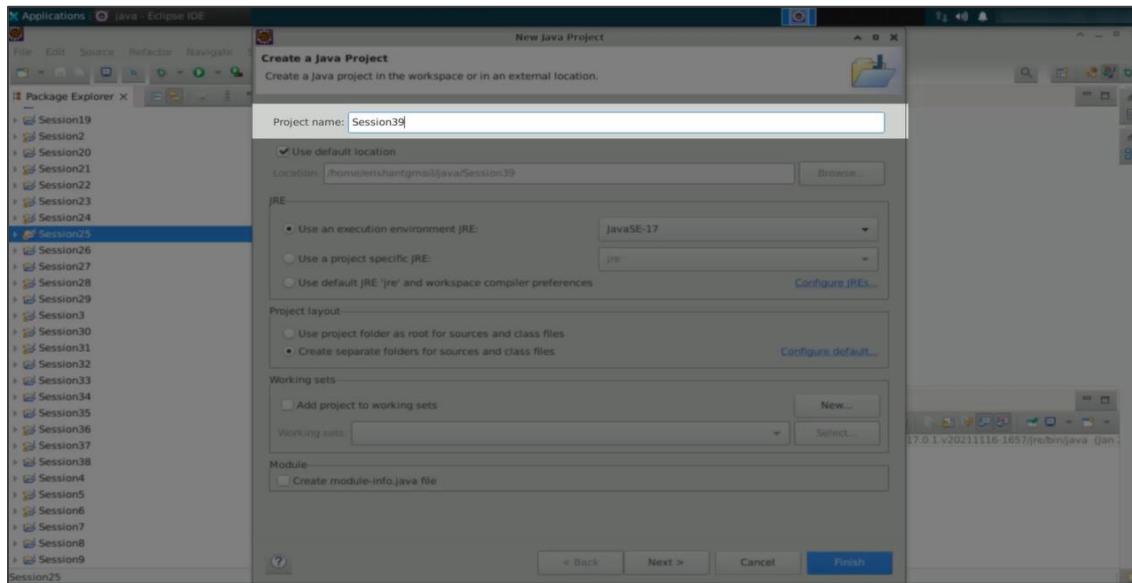
To create a HashMap, a LinkedHashMap, and a Hashtable, open the **Eclipse IDE**.



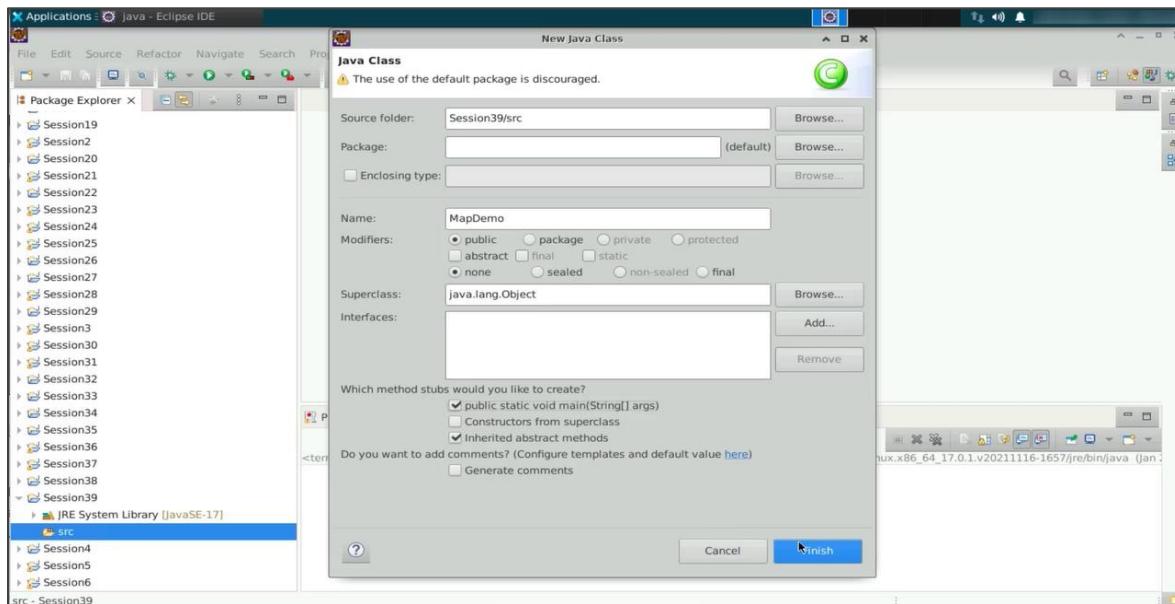
1.2 Create a new Java project



1.3 Name the new project **Session39**

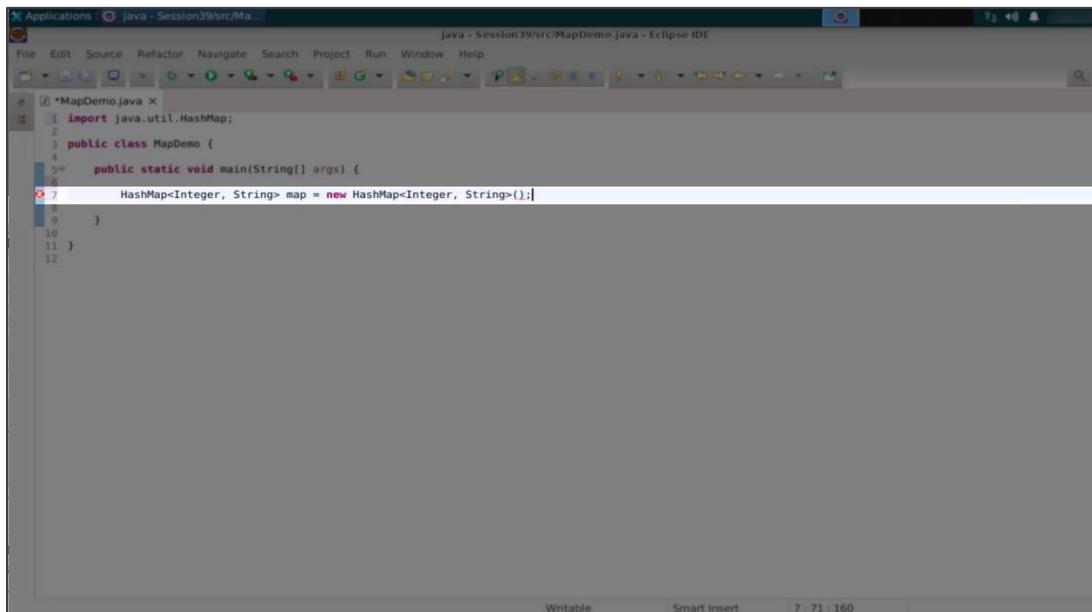


1.4 Right-click on the source folder and create a new class called **MapDemo** with the main method



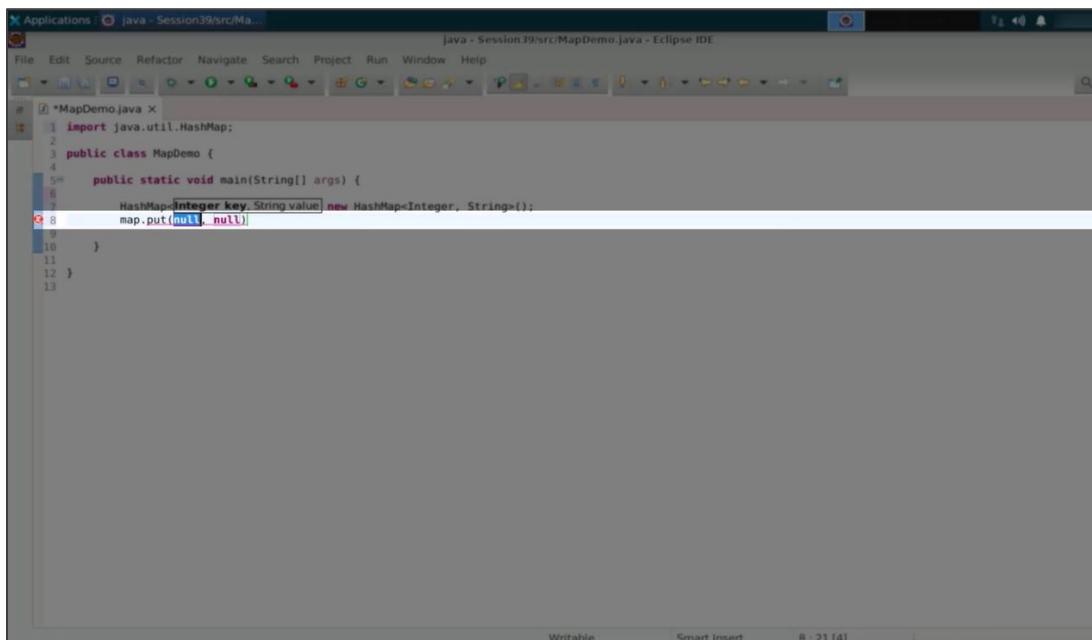
Step 2: Use a HashMap

2.1 Create a new HashMap called map



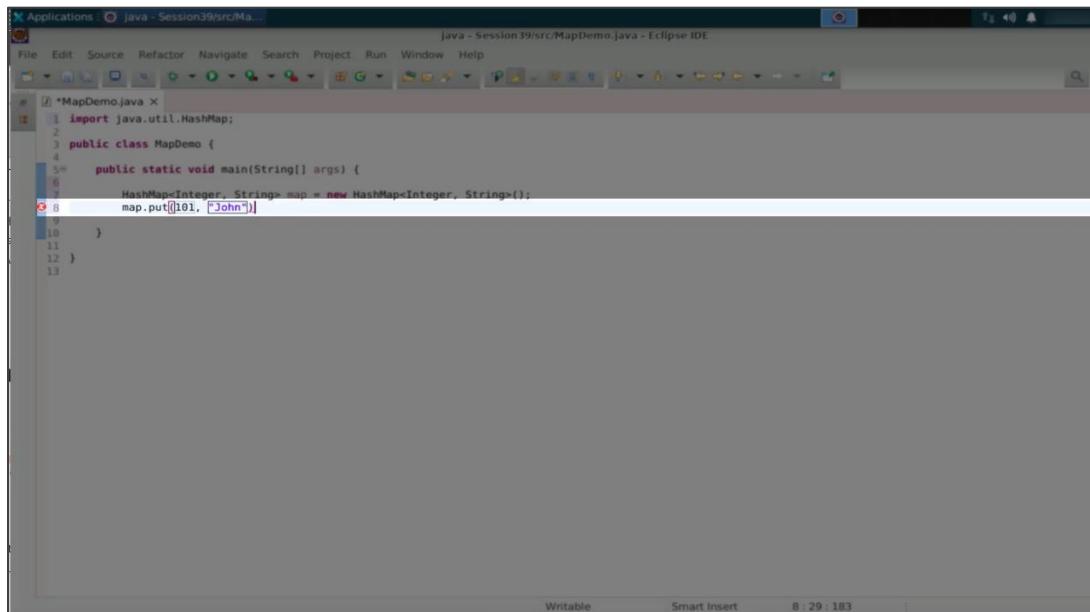
The screenshot shows the Eclipse IDE interface with a Java file named "MapDemo.java" open. The code creates a new HashMap:1 import java.util.HashMap;
2
3 public class MapDemo {
4
5 public static void main(String[] args) {
6
7 HashMap<Integer, String> map = new HashMap<Integer, String>();
8
9 }
10 }
11
12

2.2 In the map, instead of using the add method, we use the put method



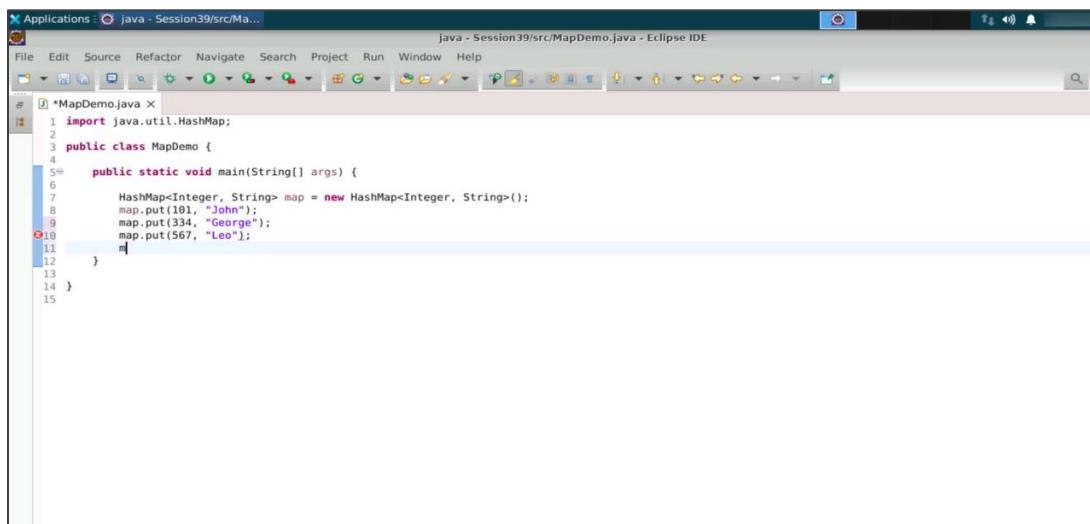
The screenshot shows the Eclipse IDE interface with the same Java file "MapDemo.java" open. The code uses the put method to add entries to the map:1 import java.util.HashMap;
2
3 public class MapDemo {
4
5 public static void main(String[] args) {
6
7 HashMap<Integer, String> map = new HashMap<Integer, String>();
8 map.put(null, null);
9
10 }
11
12 }
13

2.3 To store data inside the HashMap, use the put method. For instance, assign the key **101** and the value **John**



```
*MapDemo.java
1 import java.util.HashMap;
2
3 public class MapDemo {
4
5     public static void main(String[] args) {
6
7         HashMap<Integer, String> map = new HashMap<Integer, String>();
8         map.put(101, "John");
9
10    }
11
12 }
13
```

2.4 You can add more keys and records using the **map.put()** method

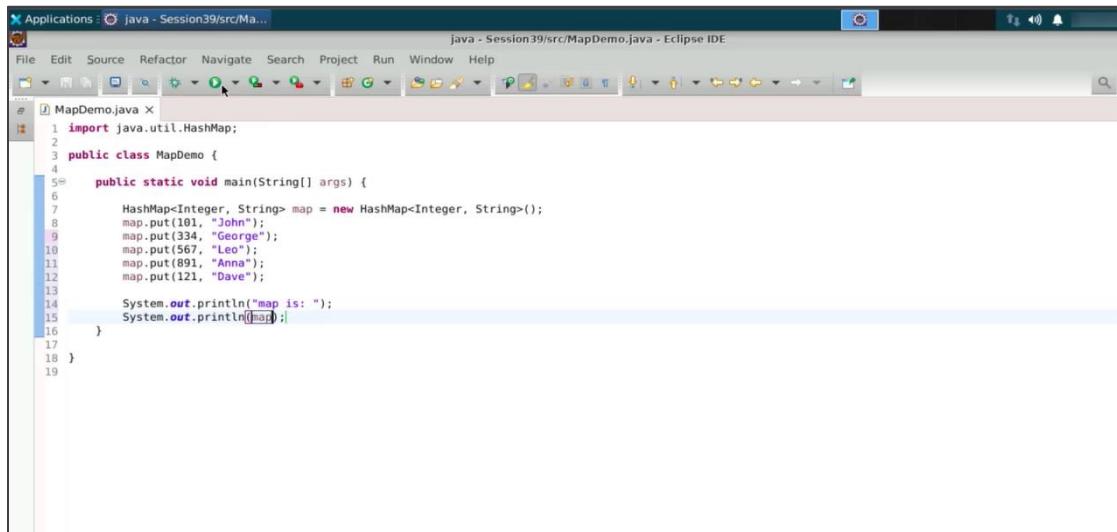


```
*MapDemo.java
1 import java.util.HashMap;
2
3 public class MapDemo {
4
5     public static void main(String[] args) {
6
7         HashMap<Integer, String> map = new HashMap<Integer, String>();
8         map.put(101, "John");
9         map.put(334, "George");
10        map.put(567, "Leo");
11    }
12
13 }
14
```

It is important to ensure that the key is unique while the value can be duplicated. Like a HashSet, a HashMap allows you to work with uniqueness based on the keys but not the values.

Step 3: Execute the code with example data

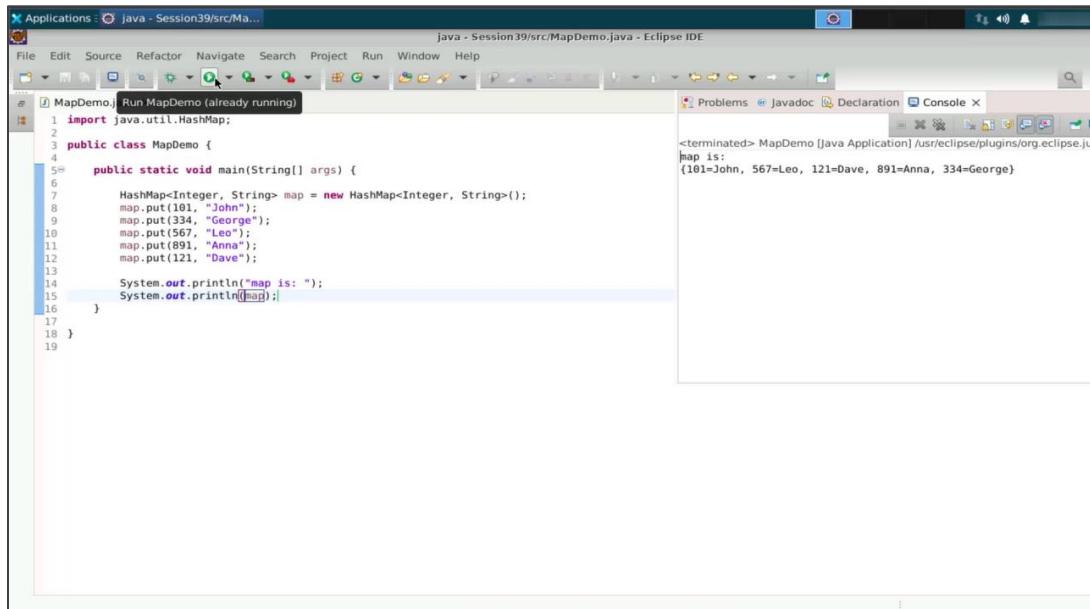
3.1 To read the data from the map, print "map is:" followed by the map itself using the print statement



The screenshot shows the Eclipse IDE interface with the title bar "Applications : java - Session 39/src/Ma...". The main window displays the Java code for "MapDemo.java". The code defines a class "MapDemo" with a main method that creates a HashMap and prints its contents using System.out.println.

```
MapDemo.java
import java.util.HashMap;
public class MapDemo {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<Integer, String>();
        map.put(101, "John");
        map.put(334, "George");
        map.put(567, "Leo");
        map.put(891, "Anna");
        map.put(121, "Dave");
        System.out.println("map is: ");
        System.out.println(map);
    }
}
```

3.2 When you run the code, you will observe the data displayed, such as **John, Leo, Dave, Anna, and George**



The screenshot shows the Eclipse IDE interface with the title bar "Applications : java - Session 39/src/Ma...". The main window displays the Java code for "MapDemo.java". The right side of the interface shows the "Console" view, which displays the output of the program execution. The output shows the string "map is:" followed by the printed map, which lists key-value pairs: 101=John, 567=Leo, 121=Dave, 891=Anna, 334=George.

```
<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.ju
map is:
{101=John, 567=Leo, 121=Dave, 891=Anna, 334=George}
```

The data is available in the form of key-value pairs.

3.3 You can add new data using the **map.put()** method. If you try to add a duplicate key, it will update the corresponding value.

The screenshot shows the Eclipse IDE interface. The title bar says "Applications : java - Session39/src/Ma...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The left pane shows the package explorer with a single file "MapDemo.java" selected. The right pane has tabs for Problems, Javadoc, Declaration, and Console. The Console tab shows the output of a run command: "<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core/bin map is: {101=John, 567=Leo, 121=Dave, 891=Anna, 334=Mike}". The code in "MapDemo.java" is:

```
1 import java.util.HashMap;
2
3 public class MapDemo {
4
5     public static void main(String[] args) {
6
7         HashMap<Integer, String> map = new HashMap<Integer, String>();
8
9         map.put(101, "John");
10        map.put(334, "George");
11        map.put(567, "Leo");
12        map.put(891, "Anna");
13        map.put(121, "Dave");
14
15        map.put(334, "Mike");
16
17        System.out.println("map is: ");
18        System.out.println(map);
19    }
20 }
21
```

3.4 You can add null as a key or a value in the HashMap

The screenshot shows the Eclipse IDE interface. The title bar says "Applications : java - Session39/src/Ma...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The left pane shows the package explorer with a single file "MapDemo.java" selected. The right pane has tabs for Problems, Javadoc, Declaration, and Console. The code in "MapDemo.java" is:

```
1 import java.util.HashMap;
2
3 public class MapDemo {
4
5     public static void main(String[] args) {
6
7         HashMap<Integer, String> map = new HashMap<Integer, String>();
8
9         map.put(101, "John");
10        map.put(334, "George");
11        map.put(567, "Leo");
12        map.put(891, "Anna");
13        map.put(121, "Dave");
14
15        // insert and update the record in the Map
16        map.put(334, "Mike");
17
18        map.put(null, "Sia");
19
20        System.out.println("map is: ");
21        System.out.println(map);
22    }
23 }
24
```

The cursor is positioned over the line "map.put(null, "Sia");".

The screenshot shows the Eclipse IDE interface with a Java file named MapDemo.java open. The code creates a HashMap and prints its contents. The output window shows the map's state.

```

java - Session39/src/Ma...
File Refactor Navigate Search Project Run Window Help
.java X
java.util.HashMap;
class MapDemo {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<Integer, String>();
        map.put(101, "John");
        map.put(102, "Sia");
        map.put(567, "Leo");
        map.put(891, "Anna");
        map.put(121, "Dave");

        // insert and update the record in the Map
        map.put(334, "Mike");
        map.put(null, "Sia");
        map.put(777, null);

        System.out.println("map is: ");
        System.out.println(map);
    }
}

```

<terminated> MapDemo [java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.hc
map is:
{null=Sia, 101=John, 567=Leo, 121=Dave, 777=null, 891=Anna, 334=Mike}

Duplicate values and multiple null values are allowed, but duplicate keys are not.

3.5 As you advance, you can use LinkedHashMap to maintain the order of insertion.

The screenshot shows the Eclipse IDE interface with a Java file named MapDemo.java open. The code creates a LinkedHashMap and prints its contents. The output window shows the map's state.

```

Applications : java - Session39/src/Ma...
File Edit Source Refactor Navigate Search Project Run Window Help
MapDemo.java X
import java.util.HashMap;
import java.util.LinkedHashMap;
public class MapDemo {
    public static void main(String[] args) {
        //HashMap<Integer, String> map = new HashMap<Integer, String>();
        LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();
        map.put(101, "John");
        map.put(334, "George");
        map.put(567, "Leo");
        map.put(891, "Anna");
        map.put(121, "Dave");

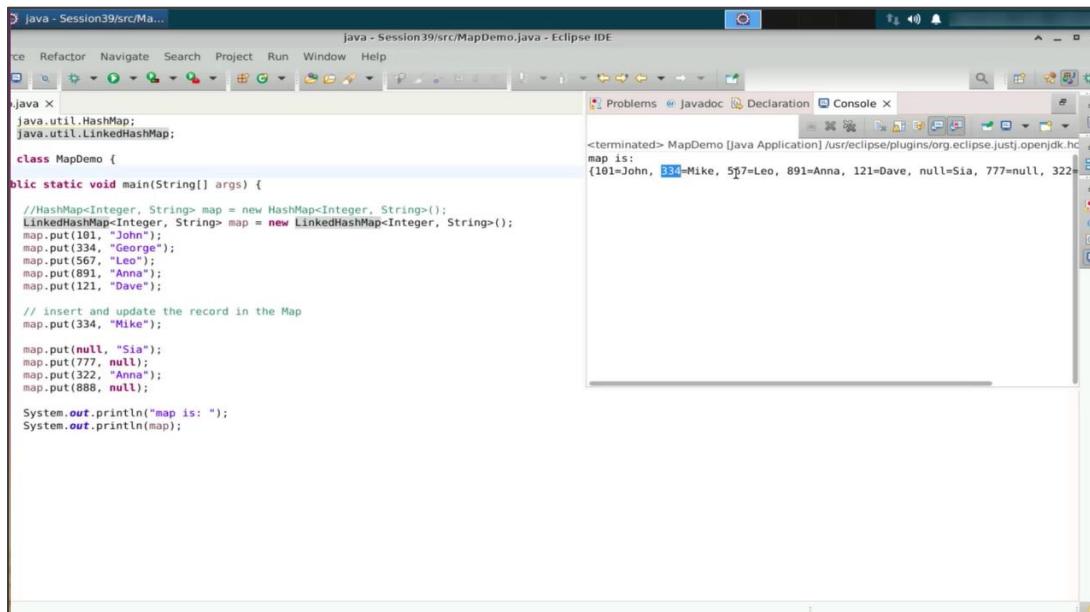
        // insert and update the record in the Map
        map.put(334, "Mike");
        map.put(null, "Sia");
        map.put(777, null);
        map.put(322, "Anna");
        map.put(888, null);

        System.out.println("map is: ");
        System.out.println(map);
    }
}

```

Writable Smart Insert 3 : 1 [47]

3.6 When you run the code, you will notice that the data is displayed in the same order in which you added it



The screenshot shows the Eclipse IDE interface with a Java file named MapDemo.java open in the editor. The code creates a LinkedHashMap of Integer and String pairs, inserting specific key-value pairs and then printing the map. The output in the Console view shows the map entries in the order they were inserted: {101=John, 334=Mike, 567=Leo, 891=Anna, 121=Dave, null=Sia, 777=null, 322=}. This demonstrates that LinkedHashMap preserves the insertion order.

```
java X
java.util.HashMap;
java.util.LinkedHashMap;
class MapDemo {
public static void main(String[] args) {
    //HashMap<Integer, String> map = new HashMap<Integer, String>();
    LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();
    map.put(101, "John");
    map.put(334, "George");
    map.put(567, "Leo");
    map.put(891, "Anna");
    map.put(121, "Dave");
    // insert and update the record in the Map
    map.put(334, "Mike");

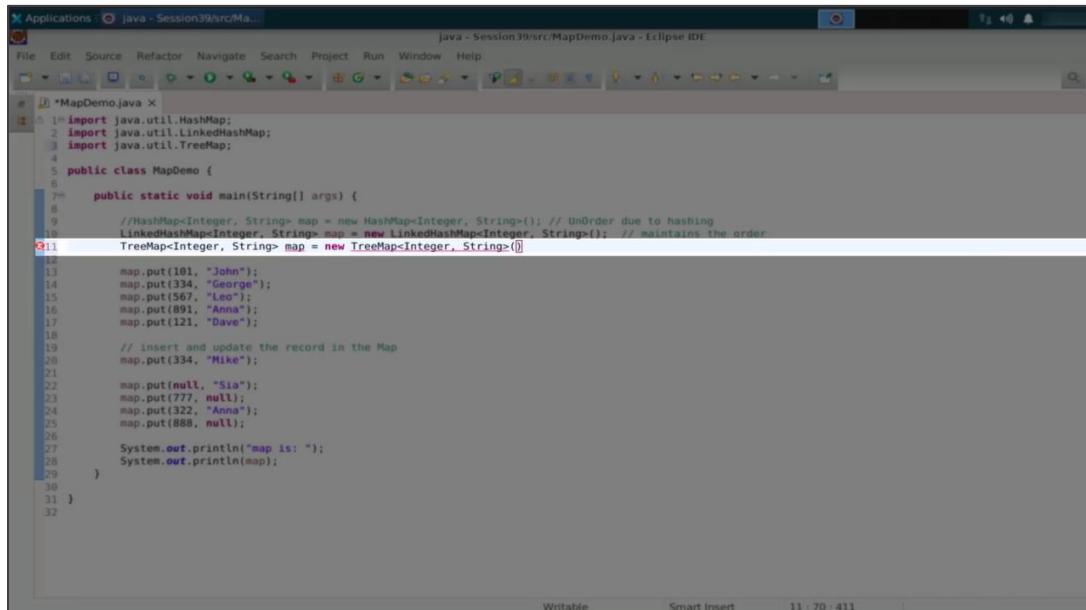
    map.put(null, "Sia");
    map.put(777, null);
    map.put(322, "Anna");
    map.put(888, null);

    System.out.println("map is: ");
    System.out.println(map);
}
```

For example, if you added the keys **101**, **334**, **567**, and **891**, the data will be shown in that exact order. This is because LinkedHashMap maintains the order of insertion.

Step 4: Sort the data based on keys

4.1 Create a new TreeMap

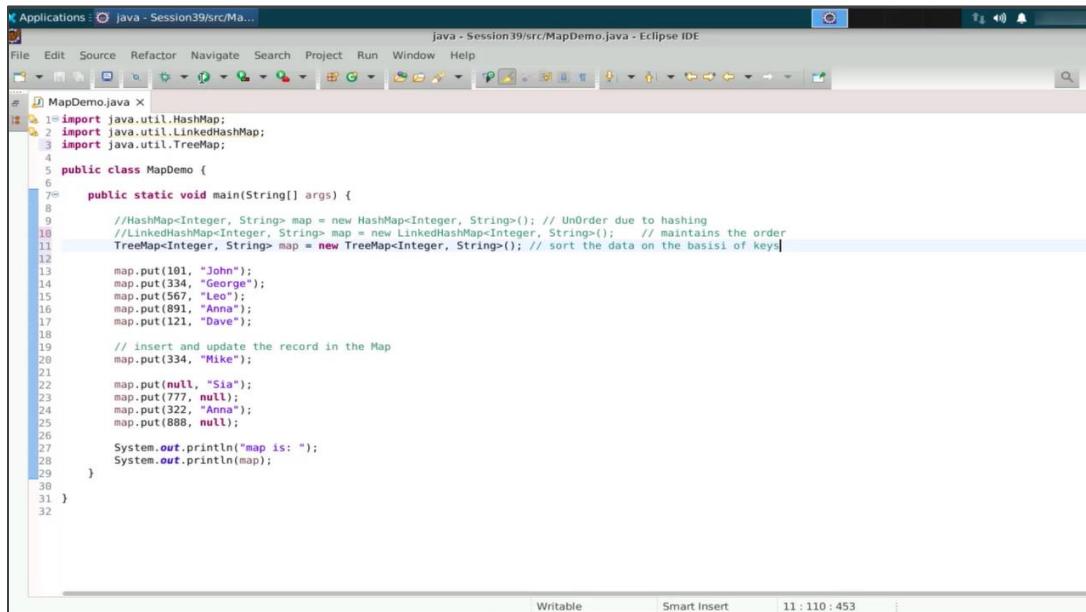


```

 1 *MapDemo.java *
 2 import java.util.HashMap;
 3 import java.util.LinkedHashMap;
 4 import java.util.TreeMap;
 5
 6 public class MapDemo {
 7
 8     public static void main(String[] args) {
 9
10         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
11         //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
12         TreeMap<Integer, String> map = new TreeMap<Integer, String>();
13
14         map.put(101, "John");
15         map.put(334, "George");
16         map.put(567, "Leo");
17         map.put(891, "Anna");
18         map.put(121, "Dave");
19
20         // insert and update the record in the Map
21         map.put(334, "Mike");
22
23         map.put(null, "Sia");
24         map.put(777, null);
25         map.put(322, "Anna");
26         map.put(888, null);
27
28         System.out.println("map is: ");
29         System.out.println(map);
30     }
31 }
32

```

4.2 Add a comment indicating that this TreeMap will sort the data based on keys



```

 1 *MapDemo.java *
 2 import java.util.HashMap;
 3 import java.util.LinkedHashMap;
 4 import java.util.TreeMap;
 5
 6 public class MapDemo {
 7
 8     public static void main(String[] args) {
 9
10         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
11         //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
12         TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
13
14         map.put(101, "John");
15         map.put(334, "George");
16         map.put(567, "Leo");
17         map.put(891, "Anna");
18         map.put(121, "Dave");
19
20         // insert and update the record in the Map
21         map.put(334, "Mike");
22
23         map.put(null, "Sia");
24         map.put(777, null);
25         map.put(322, "Anna");
26         map.put(888, null);
27
28         System.out.println("map is: ");
29         System.out.println(map);
30     }
31 }
32

```

4.3 Resolve the null key issue

The screenshot shows the Eclipse IDE interface. The code editor displays a Java file named MapDemo.java. The code creates a TreeMap and inserts several entries, including a null key entry. The console tab shows the output of the program, which ends with a stack trace of a NullPointerException:

```

<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hc
Exception in thread "main" java.lang.NullPointerException
at java.base/java.util.Objects.requireNonNull(Objects.java:208)
at java.base/java.util.TreeMap.put(TreeMap.java:899)
at java.base/java.util.TreeMap.put(TreeMap.java:534)
at MapDemo.main(MapDemo.java:22)

```

4.4 Run the code and observe the sorted data based on keys

The screenshot shows the Eclipse IDE interface. The code editor displays the same MapDemo.java file. The console tab shows the output of the program, which prints the sorted map entries:

```

map is:
{101=John, 121=Dave, 322=Anna, 334=Mike, 567=Leo, 777=null, 81=null}

```

Step 5: Iterate the data structure with example data

5.1 Execute built-in methods on the map such as `map.get(101)`

```

Applications Java - Session39/src/Ma...
File Edit Source Refactor Navigate Search Project Run Window Help
*MapDemo.java X
1 import java.util.HashMap;
2 import java.util.LinkedHashMap;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        map.put(101, "John");
14        map.put(334, "George");
15        map.put(567, "Leo");
16        map.put(891, "Anna");
17        map.put(121, "Dave");
18
19        // insert and update the record in the Map
20        map.put(334, "Mike");
21
22        //map.put(null, "Sia");
23        map.put(777, null);
24        map.put(322, "Anna");
25        map.put(888, null);
26
27        System.out.println("map is: ");
28        System.out.println(map);
29
30        I map.get(101);
31
32    }
33
34 }
35

```

Writable Smart Insert 30 : 22 : 829

5.2 Assign the retrieved value to a variable named name: (`String name = map.get(101)`)

```

Applications Java - Session39/src/Ma...
File Edit Source Refactor Navigate Search Project Run Window Help
*MapDemo.java X
1 import java.util.HashMap;
2 import java.util.LinkedHashMap;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        map.put(101, "John");
14        map.put(334, "George");
15        map.put(567, "Leo");
16        map.put(891, "Anna");
17        map.put(121, "Dave");
18
19        // insert and update the record in the Map
20        map.put(334, "Mike");
21
22        //map.put(null, "Sia");
23        map.put(777, null);
24        map.put(322, "Anna");
25        map.put(888, null);
26
27        System.out.println("map is: ");
28        System.out.println(map);
29
30        String name = map.get(101);
31
32    }
33
34 }
35

```

Writable Smart Insert 30 : 22 : 829

5.3 Print the value of name: ("name is: " + name)

The screenshot shows the Eclipse IDE interface with the title bar "Applications : java - Session39/src/Ma...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left sidebar shows the project structure with "MapDemo.java" selected. The main editor area contains the following Java code:

```
1 import java.util.HashMap;
2 import java.util.LinkedHashMap;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        map.put(101, "John");
14        map.put(334, "George");
15        map.put(567, "Leo");
16        map.put(891, "Anna");
17        map.put(121, "Dave");
18
19        // insert and update the record in the Map
20        map.put(334, "Mike");
21
22        //map.put(null, "Sia");
23        map.put(777, null);
24        map.put(322, "Anna");
25        map.put(888, null);
26
27        System.out.println("map is: ");
28        System.out.println(map);
29
30        String name = map.get(101);
31        System.out.println("name is: "+name);
32    }
33}
34
35 }
```

5.4 Run the code and observe the printed name

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** java - Session39/src/Ma... and java - Session39/src/MapDemo.java - Eclipse IDE.
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Left Panel (Code View):** The code for `MapDemo.java` is displayed. It imports `java.util.HashMap`, `java.util.LinkedHashMap`, and `java.util TreeMap`. The class `MapDemo` contains a static main method that creates three different map types: `HashMap`, `LinkedHashMap`, and `TreeMap`. It then inserts key-value pairs and prints them out. The code uses `System.out.println` statements to output the maps and specific names.
- Right Panel (Console View):** The output of the application is shown in the Console tab. It includes the terminated message, the map types, and the printed data. The printed data shows a map where the key is the name and the value is the ID. The output ends with the message "name is: John".

```
java - Session39/src/Ma...
java - Session39/src/MapDemo.java - Eclipse IDE

File Refactor Navigate Search Project Run Window Help

src.java X
Problems Javadoc Declaration Console X

<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.openjdk.h
map is:
{101=John, 121=Dave, 322=Anna, 334=Mike, 567=Leo, 777=null, 888=null, 891=
name is: John

java.util.HashMap;
java.util.LinkedHashMap;
java.util.TreeMap;

class MapDemo {
    public static void main(String[] args) {
        //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to h
        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // m
        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on t

        map.put(101, "John");
        map.put(334, "George");
        map.put(567, "Leo");
        map.put(891, "Anna");
        map.put(121, "Dave");

        // insert and update the record in the Map
        map.put(334, "Mike");

        //map.put(null, "Sia");
        map.put(777, null);
        map.put(322, "Anna");
        map.put(888, null);

        System.out.println("map is: ");
        System.out.println(map);

        String name = map.get(101);
        System.out.println("name is: "+name);
    }
}
```

5.5 Repeat steps 9.1-9.4 with different keys to retrieve corresponding values

The screenshot shows the Eclipse IDE interface with the Java - Session 39/src/MapDemo.java file open. The code demonstrates how to use a Map to store names and IDs. The output window shows the map's contents and a retrieval operation.

```

1 import java.util.HashMap;
2 import java.util.LinkedHashMap;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        map.put(101, "John");
14        map.put(334, "George");
15        map.put(567, "Leo");
16        map.put(891, "Anna");
17        map.put(121, "Dave");
18
19        // insert and update the record in the Map
20        map.put(334, "Mike");
21
22        //map.put(null, "Sia");
23        map.put(777, null);
24        map.put(322, "Anna");
25        map.put(888, null);
26
27        System.out.println("map is: ");
28        System.out.println(map);
29
30        String name = map.get(121);
31        System.out.println("name is: "+name);
32
33    }
34
35 }
36

```

5.6 Use `map.containsKey(567)` to check if the key 567 exists

The screenshot shows the Eclipse IDE interface with the Java - Session 39/src/MapDemo.java file open. A conditional statement has been added to the code to check if the key 567 exists in the map.

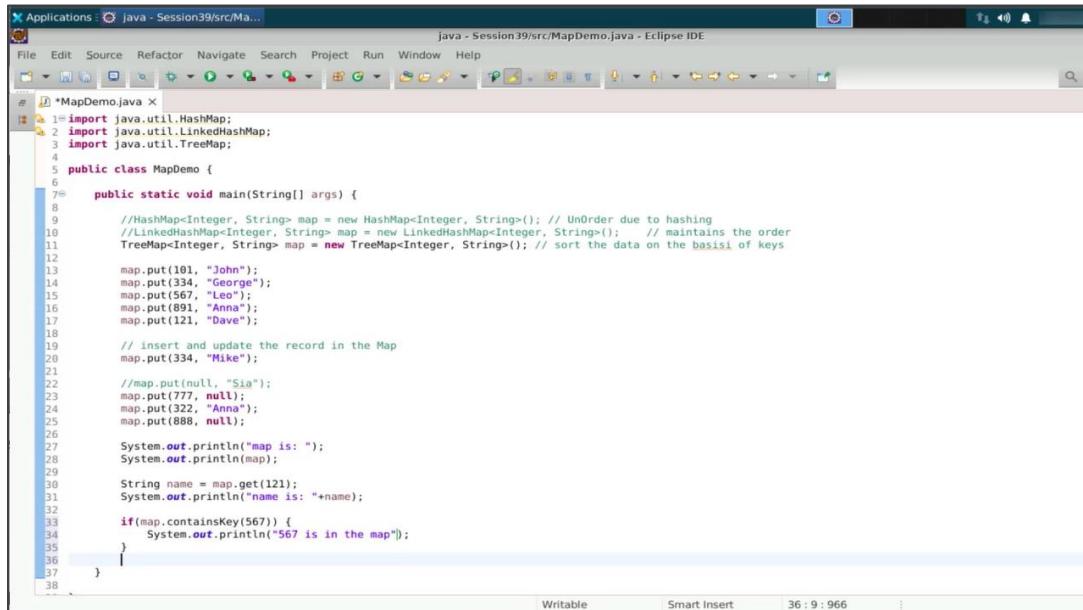
```

1 import java.util.HashMap;
2 import java.util.LinkedHashMap;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        map.put(101, "John");
14        map.put(334, "George");
15        map.put(567, "Leo");
16        map.put(891, "Anna");
17        map.put(121, "Dave");
18
19        // insert and update the record in the Map
20        map.put(334, "Mike");
21
22        //map.put(null, "Sia");
23        map.put(777, null);
24        map.put(322, "Anna");
25        map.put(888, null);
26
27        System.out.println("map is: ");
28        System.out.println(map);
29
30        String name = map.get(121);
31        System.out.println("name is: "+name);
32
33        if(map.containsKey(567))
34
35    }
36

```

A syntax error message is visible at the bottom left: "Syntax error on token ")", Statement expected after this token".

5.7 Print 567 is in the map if the key is present

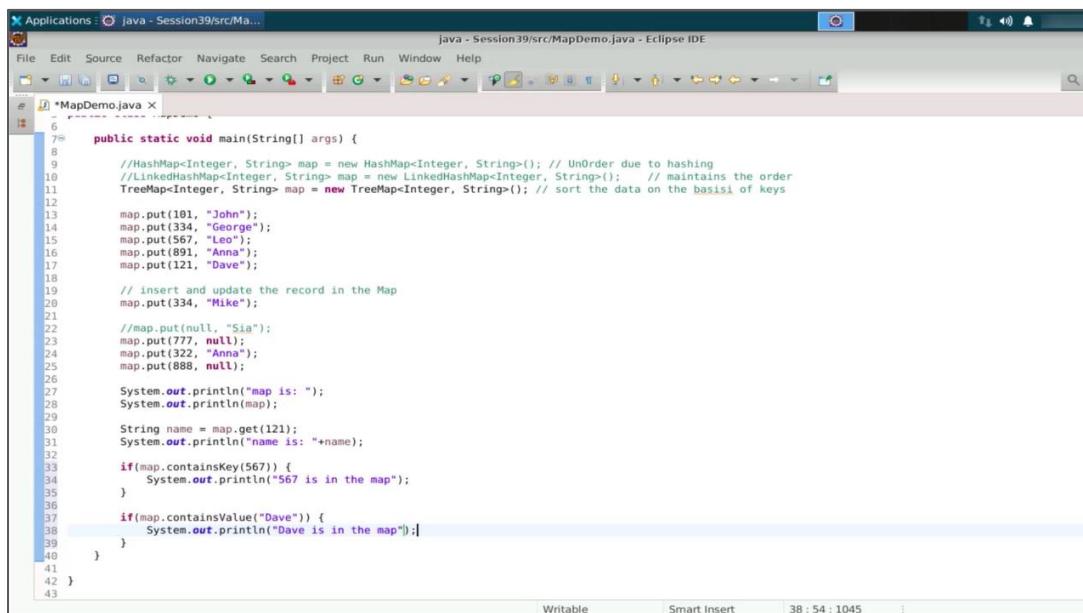


```

1 *MapDemo.java X
2
3 import java.util.HashMap;
4 import java.util.LinkedHashMap;
5 import java.util.TreeMap;
6
7 public class MapDemo {
8
9     public static void main(String[] args) {
10
11         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
12         //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
13         TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
14
15         map.put(101, "John");
16         map.put(334, "George");
17         map.put(567, "Leo");
18         map.put(891, "Anna");
19         map.put(121, "Dave");
20
21         // insert and update the record in the Map
22         map.put(334, "Mike");
23
24         //map.put(null, "Sia");
25         map.put(777, null);
26         map.put(322, "Anna");
27         map.put(888, null);
28
29         System.out.println("map is: ");
30         System.out.println(map);
31
32         String name = map.get(121);
33         System.out.println("name is: "+name);
34
35         if(map.containsKey(567)) {
36             System.out.println("567 is in the map");
37         }
38     }
39 }

```

5.8 Use map.containsValue("Dave") to check if the value Dave exists



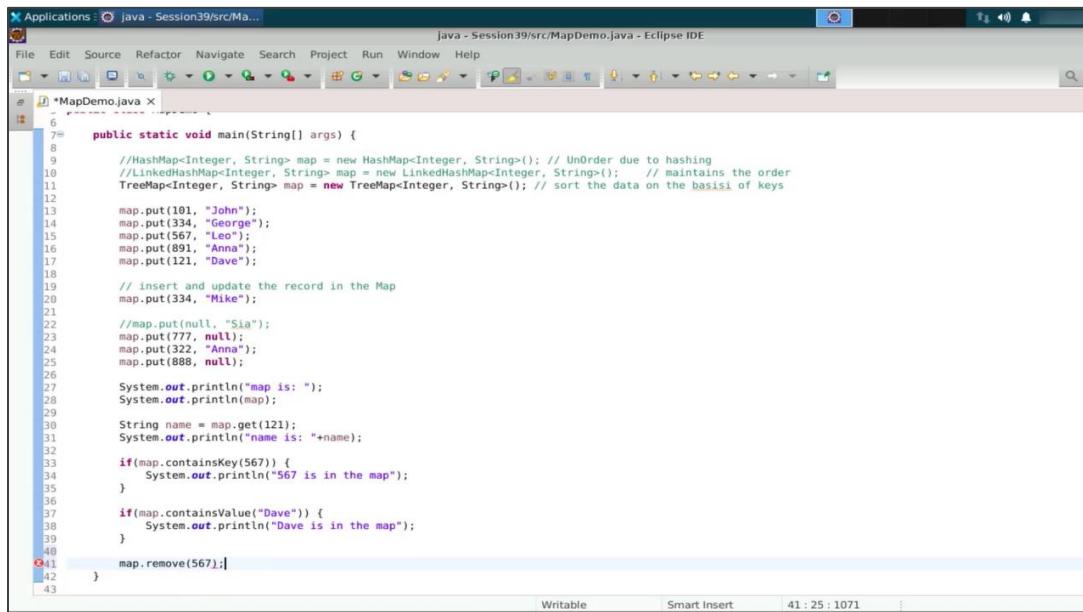
```

1 *MapDemo.java X
2
3 import java.util.HashMap;
4 import java.util.LinkedHashMap;
5 import java.util.TreeMap;
6
7 public class MapDemo {
8
9     public static void main(String[] args) {
10
11         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
12         //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
13         TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
14
15         map.put(101, "John");
16         map.put(334, "George");
17         map.put(567, "Leo");
18         map.put(891, "Anna");
19         map.put(121, "Dave");
20
21         // insert and update the record in the Map
22         map.put(334, "Mike");
23
24         //map.put(null, "Sia");
25         map.put(777, null);
26         map.put(322, "Anna");
27         map.put(888, null);
28
29         System.out.println("map is: ");
30         System.out.println(map);
31
32         String name = map.get(121);
33         System.out.println("name is: "+name);
34
35         if(map.containsKey(567)) {
36             System.out.println("567 is in the map");
37         }
38
39         if(map.containsValue("Dave")) {
40             System.out.println("Dave is in the map");
41         }
42     }
43 }

```

Step 6: Use the remove() method and execute the code

6.1 Use map.remove(567) to remove a key-value pair



```

public static void main(String[] args) {
    //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
    //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
    TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys

    map.put(101, "John");
    map.put(334, "George");
    map.put(567, "Leo");
    map.put(891, "Anna");
    map.put(121, "Dave");

    // insert and update the record in the Map
    map.put(334, "Mike");

    //map.put(null, "Sia");
    map.put(777, null);
    map.put(322, "Anna");
    map.put(888, null);

    System.out.println("map is: ");
    System.out.println(map);

    String name = map.get(121);
    System.out.println("name is: "+name);

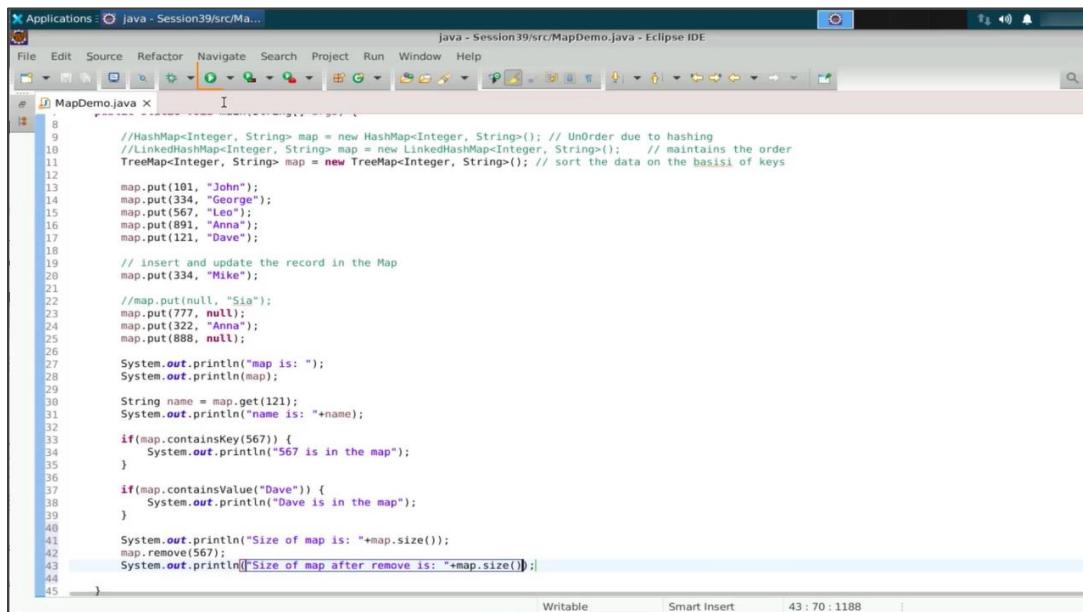
    if(map.containsKey(567)) {
        System.out.println("567 is in the map");
    }

    if(map.containsValue("Dave")) {
        System.out.println("Dave is in the map");
    }

    map.remove(567);
}

```

6.2 Print the size of the map before and after using remove():



```

System.out.println("Size of map is: "+map.size());
map.remove(567);
System.out.println("Size of map after remove is: "+map.size());

```

6.3 Run the code and observe the size change

```

8
9     //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to h
10    //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // m
11    TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on k
12
13    map.put(101, "John");
14    map.put(334, "George");
15    map.put(567, "Leo");
16    map.put(891, "Anna");
17    map.put(121, "Dave");
18
19    // insert and update the record in the Map
20    map.put(334, "Mike");
21
22    //map.put(null, "Sia");
23    map.put(777, null);
24    map.put(322, "Anna");
25    map.put(888, null);
26
27    System.out.println("map is: ");
28    System.out.println(map);
29
30    String name = map.get(121);
31    System.out.println("name is: "+name);
32
33    if(map.containsKey(567)) {
34        System.out.println("567 is in the map");
35    }
36
37    if(map.containsValue("Dave")) {
38        System.out.println("Dave is in the map");
39    }
40
41    System.out.println("Size of map is: "+map.size());
42    map.remove(567);
43    System.out.println("Size of map after remove is: "+map.size());
44
45
46
47
48
49

```

Step 7: Implement iteration to obtain all the keys from the map

7.1 Obtain all the keys from the map using `map.keySet()` and assign the keys to a set:

`Set<Integer> keys = map.keySet();`

```

12    TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
13
14    map.put(101, "John");
15    map.put(334, "George");
16    map.put(567, "Leo");
17    map.put(891, "Anna");
18    map.put(121, "Dave");
19
20    // insert and update the record in the Map
21    map.put(334, "Mike");
22
23    //map.put(null, "Sia");
24    map.put(777, null);
25    map.put(322, "Anna");
26    map.put(888, null);
27
28    System.out.println("map is: ");
29    System.out.println(map);
30
31    String name = map.get(121);
32    System.out.println("name is: "+name);
33
34    if(map.containsKey(567)) {
35        System.out.println("567 is in the map");
36    }
37
38    if(map.containsValue("Dave")) {
39        System.out.println("Dave is in the map");
40    }
41
42    System.out.println("Size of map is: "+map.size());
43    map.remove(567);
44    System.out.println("Size of map after remove is: "+map.size());
45
46    Set<Integer> keys = map.keySet();
47
48
49

```

7.2 Print the keys: ("Keys in the map are: " + keys);

```

 1 * Applications  java - Session39/src/Ma...
 2 
 3 File Edit Source Refactor Navigate Search Project Run Window Help
 4 
 5 *MapDemo.java X
 6 
 7 TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
 8 
 9 map.put(101, "John");
10 map.put(334, "George");
11 map.put(567, "Leo");
12 map.put(891, "Anna");
13 map.put(121, "Dave");
14 
15 // insert and update the record in the Map
16 map.put(334, "Mike");
17 
18 //map.put(null, "Sia");
19 map.put(777, null);
20 map.put(322, "Anna");
21 map.put(888, null);
22 
23 System.out.println("map is: ");
24 System.out.println(map);
25 
26 String name = map.get(121);
27 System.out.println("name is: "+name);
28 
29 if(map.containsKey(567)) {
30     System.out.println("567 is in the map");
31 }
32 
33 if(map.containsValue("Dave")) {
34     System.out.println("Dave is in the map");
35 }
36 
37 System.out.println("Size of map is: "+map.size());
38 map.remove(567);
39 System.out.println("Size of map after remove is: "+map.size());
40 
41 Set<Integer> keys = map.keySet();
42 System.out.println("Keys in map are: "+keys);
43 
44 }
45 
```

The code creates a TreeMap and adds several entries. It then prints the map, checks for specific keys and values, and finally prints the set of keys.

7.3 Run the code and observe the printed keys

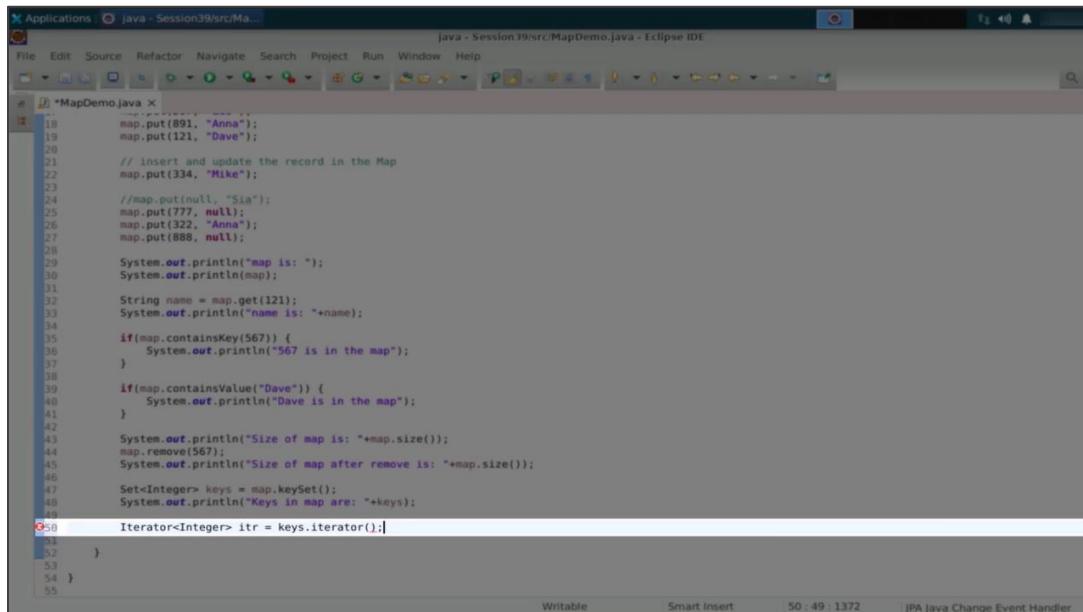
The screenshot shows the Eclipse IDE with the 'Run MapDemo' configuration selected. The console output window shows the execution results:

```

<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core
map is:
[101=John, 121=Dave, 322=Anna, 334=Mike, 567=Leo, 777=null, 888=null]
name is: Dave
567 is in the map
Dave is in the map
Size of map is: 8
Size of map after remove is: 7
Keys in map are: [101, 121, 322, 334, 777, 888, 891]

```

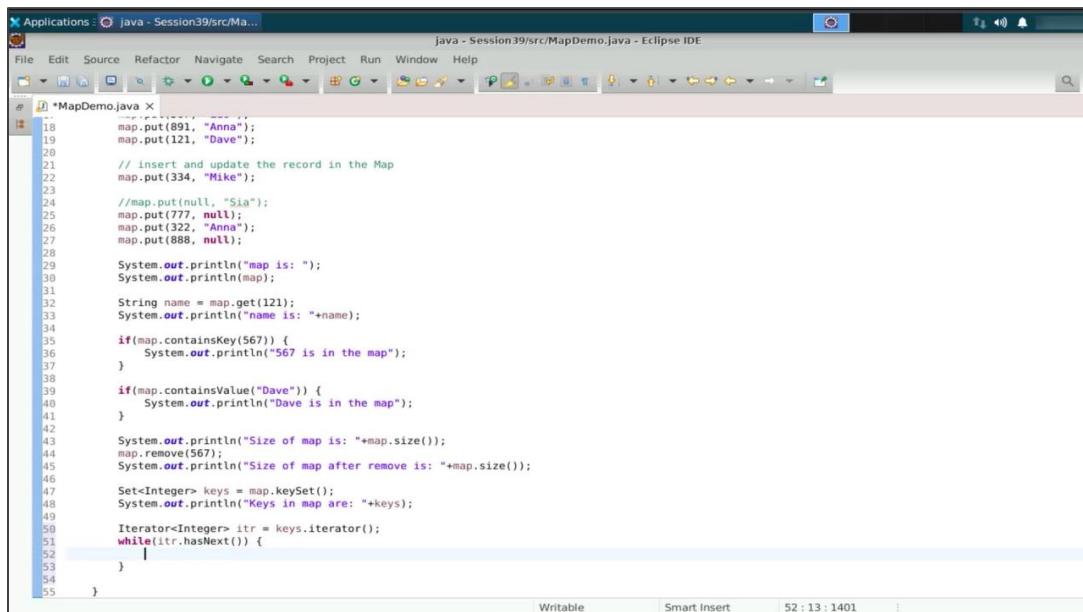
7.4 Create an iterator for the keys: `Iterator<Integer> itr = keys.iterator();`



```

 1 *MapDemo.java X
 2
 3 import java.util.*;
 4
 5 public class MapDemo {
 6     public static void main(String[] args) {
 7         // Create a map
 8         Map<Integer, String> map = new HashMap<Integer, String>();
 9
10         // insert and update the record in the Map
11         map.put(891, "Anna");
12         map.put(121, "Dave");
13
14         // insert and update the record in the Map
15         map.put(334, "Mike");
16
17         //map.put(null, "Sia");
18         map.put(777, null);
19         map.put(322, "Anna");
20         map.put(888, null);
21
22         System.out.println("map is: ");
23         System.out.println(map);
24
25         String name = map.get(121);
26         System.out.println("name is: "+name);
27
28         if(map.containsKey(567)) {
29             System.out.println("567 is in the map");
30         }
31
32         if(map.containsValue("Dave")) {
33             System.out.println("Dave is in the map");
34         }
35
36         System.out.println("Size of map is: "+map.size());
37         map.remove(567);
38         System.out.println("Size of map after remove is: "+map.size());
39
40         Set<Integer> keys = map.keySet();
41         System.out.println("Keys in map are: "+keys);
42
43         Iterator<Integer> itr = keys.iterator();
44
45     }
46 }
47
48 }
```

7.5 Use a while loop with `itr.hasNext()` to iterate over the keys



```

 1 *MapDemo.java X
 2
 3 import java.util.*;
 4
 5 public class MapDemo {
 6     public static void main(String[] args) {
 7         // Create a map
 8         Map<Integer, String> map = new HashMap<Integer, String>();
 9
10         // insert and update the record in the Map
11         map.put(891, "Anna");
12         map.put(121, "Dave");
13
14         // insert and update the record in the Map
15         map.put(334, "Mike");
16
17         //map.put(null, "Sia");
18         map.put(777, null);
19         map.put(322, "Anna");
20         map.put(888, null);
21
22         System.out.println("map is: ");
23         System.out.println(map);
24
25         String name = map.get(121);
26         System.out.println("name is: "+name);
27
28         if(map.containsKey(567)) {
29             System.out.println("567 is in the map");
30         }
31
32         if(map.containsValue("Dave")) {
33             System.out.println("Dave is in the map");
34         }
35
36         System.out.println("Size of map is: "+map.size());
37         map.remove(567);
38         System.out.println("Size of map after remove is: "+map.size());
39
40         Set<Integer> keys = map.keySet();
41         System.out.println("Keys in map are: "+keys);
42
43         Iterator<Integer> itr = keys.iterator();
44         while(itr.hasNext()) {
45             |
46         }
47
48     }
49 }
50
51 }
```

7.6 Retrieve the corresponding value using map.get()

The screenshot shows the Eclipse IDE interface with the title bar "Java - Session39/src/MapDemo.java - Eclipse IDE". The code editor displays Java code for a MapDemo class. At line 53, there is a call to `map.get(key)`. A tooltip from the IDE's code completion feature is visible, showing the method signature `Object get(Object key)`. The code itself contains various map operations like put and get.

```

18     map.put(891, "Anna");
19     map.put(121, "Dave");
20
21     // insert and update the record in the Map
22     map.put(334, "Mike");
23
24     //map.put(null, "Sia");
25     map.put(777, null);
26     map.put(322, "Anna");
27     map.put(888, null);
28
29     System.out.println("map is: ");
30     System.out.println(map);
31
32     String name = map.get(121);
33     System.out.println("name is: "+name);
34
35     if(map.containsKey(567)) {
36         System.out.println("567 is in the map");
37     }
38
39     if(map.containsValue("Dave")) {
40         System.out.println("Dave is in the map");
41     }
42
43     System.out.println("Size of map is: "+map.size());
44     map.remove(567);
45     System.out.println("Size of map after remove is: "+map.size());
46
47     Set<Integer> keys = map.keySet();
48     System.out.println("Keys in map are: "+keys);
49
50     Iterator<Integer> itr = keys.iterator();
51     while(itr.hasNext()) {
52         Integer key = itr.next();
53         String value = map.get(key);
54     }
55

```

7.7 Print the key-value pairs

The screenshot shows the Eclipse IDE interface with the title bar "Java - Session39/src/MapDemo.java - Eclipse IDE". The code editor displays Java code for a MapDemo class. At line 54, there is a call to `System.out.println(key + " " + value);`. A tooltip from the IDE's code completion feature is visible, showing the method signature `void println(String x)`. The code itself contains various map operations like put and get.

```

22     map.put(334, "Mike");
23
24     //map.put(null, "Sia");
25     map.put(777, null);
26     map.put(322, "Anna");
27     map.put(888, null);
28
29     System.out.println("map is: ");
30     System.out.println(map);
31
32     String name = map.get(121);
33     System.out.println("name is: "+name);
34
35     if(map.containsKey(567)) {
36         System.out.println("567 is in the map");
37     }
38
39     if(map.containsValue("Dave")) {
40         System.out.println("Dave is in the map");
41     }
42
43     System.out.println("Size of map is: "+map.size());
44     map.remove(567);
45     System.out.println("Size of map after remove is: "+map.size());
46
47     Set<Integer> keys = map.keySet();
48     System.out.println("Keys in map are: "+keys);
49
50     Iterator<Integer> itr = keys.iterator();
51     while(itr.hasNext()) {
52         Integer key = itr.next();
53         String value = map.get(key);
54         System.out.println(key + " " + value);
55     }
56
57
58

```

7.8 Run the code and observe the printed key-value pairs

The screenshot shows the Eclipse IDE interface with the Java perspective selected. A Java file named `MapDemo.java` is open in the editor. The code demonstrates various operations on a map, including putting key-value pairs, checking for keys and values, and removing elements. The output window shows the results of the program's execution, including the initial state of the map, the removal of key 567, and the final size of the map.

```

    1  package com.simplilearn;
    2
    3  import java.util.*;
    4
    5  public class MapDemo {
    6
    7      public static void main(String[] args) {
    8
    9          Map<Integer, String> map = new HashMap<Integer, String>();
   10
   11          map.put(101, "John");
   12          map.put(121, "Dave");
   13          map.put(322, "Anna");
   14          map.put(334, "Mike");
   15          map.put(567, "Leo");
   16          map.put(777, null);
   17          map.put(888, null);
   18
   19          System.out.println("map is: ");
   20          System.out.println(map);
   21
   22          String name = map.get(121);
   23          System.out.println("name is: "+name);
   24
   25          if(map.containsKey(567)) {
   26              System.out.println("567 is in the map");
   27          }
   28
   29          if(map.containsValue("Dave")) {
   30              System.out.println("Dave is in the map");
   31          }
   32
   33          System.out.println("Size of map is: "+map.size());
   34          map.remove(567);
   35          System.out.println("Size of map after remove is: "+map.size());
   36
   37          Set<Integer> keys = map.keySet();
   38          System.out.println("Keys in map are: "+keys);
   39
   40          Iterator<Integer> itr = keys.iterator();
   41          while(itr.hasNext()) {
   42              Integer key = itr.next();
   43              String value = map.get(key);
   44
   45              System.out.println(key+" \t "+value);
   46          }
   47      }
   48  }

```

Step 8: Execute the entrySet() method and iterate through the code

8.1 Obtain the entry set using `map.entrySet()`

The screenshot shows the Eclipse IDE interface with the Java perspective selected. A Java file named `MapDemo.java` is open in the editor. The code is identical to the previous screenshot, but the line `Set<Entry<Integer, String>> set = map.entrySet();` is highlighted in blue, indicating it is the current point of interest.

```

    1  package com.simplilearn;
    2
    3  import java.util.*;
    4
    5  public class MapDemo {
    6
    7      public static void main(String[] args) {
    8
    9          Map<Integer, String> map = new HashMap<Integer, String>();
   10
   11          map.put(101, "John");
   12          map.put(121, "Dave");
   13          map.put(322, "Anna");
   14          map.put(334, "Mike");
   15          map.put(567, "Leo");
   16          map.put(777, null);
   17          map.put(888, null);
   18
   19          System.out.println("map is: ");
   20          System.out.println(map);
   21
   22          String name = map.get(121);
   23          System.out.println("name is: "+name);
   24
   25          if(map.containsKey(567)) {
   26              System.out.println("567 is in the map");
   27          }
   28
   29          if(map.containsValue("Dave")) {
   30              System.out.println("Dave is in the map");
   31          }
   32
   33          System.out.println("Size of map is: "+map.size());
   34          map.remove(567);
   35          System.out.println("Size of map after remove is: "+map.size());
   36
   37          Set<Integer> keys = map.keySet();
   38          System.out.println("Keys in map are: "+keys);
   39
   40          Iterator<Integer> itr = keys.iterator();
   41          while(itr.hasNext()) {
   42              Integer key = itr.next();
   43              String value = map.get(key);
   44
   45              System.out.println(key+" \t "+value);
   46          }
   47
   48          Set<Entry<Integer, String>> set = map.entrySet();
   49      }
   50  }

```

8.2 Assign the entry set to a variable:

```
Set<Map.Entry<Integer, String>> entrySet = map.entrySet();
```

```

    27     map.put(322, "Anna");
    28     map.put(888, null);
    29
    30     System.out.println("map is: ");
    31     System.out.println(map);
    32
    33     String name = map.get(121);
    34     System.out.println("name is: "+name);
    35
    36     if(map.containsKey(567)) {
    37         System.out.println("567 is in the map");
    38     }
    39
    40     if(map.containsValue("Dave")) {
    41         System.out.println("Dave is in the map");
    42     }
    43
    44     System.out.println("Size of map is: "+map.size());
    45     map.remove(567);
    46     System.out.println("Size of map after remove is: "+map.size());
    47
    48     Set<Integer> keys = map.keySet();
    49     System.out.println("Keys in map are: "+keys);
    50
    51     Iterator<Integer> itr = keys.iterator();
    52     while(itr.hasNext()) {
    53         Integer key = itr.next();
    54         String value = map.get(key);
    55
    56         System.out.println(key+" \t "+value);
    57     }
    58
    59     Set<Entry<Integer, String>> set = map.entrySet();
    60     Iterator<Entry<Integer, String>> itr1 = set.iterator();
    61
    62 }
    63
    64 }
```

8.3 Create an iterator for the entry set:

```
Iterator<Map.Entry<Integer, String>> itr1 = entrySet.iterator();
```

```

    27     map.put(322, "Anna");
    28     map.put(888, null);
    29
    30     System.out.println("map is: ");
    31     System.out.println(map);
    32
    33     String name = map.get(121);
    34     System.out.println("name is: "+name);
    35
    36     if(map.containsKey(567)) {
    37         System.out.println("567 is in the map");
    38     }
    39
    40     if(map.containsValue("Dave")) {
    41         System.out.println("Dave is in the map");
    42     }
    43
    44     System.out.println("Size of map is: "+map.size());
    45     map.remove(567);
    46     System.out.println("Size of map after remove is: "+map.size());
    47
    48     Set<Integer> keys = map.keySet();
    49     System.out.println("Keys in map are: "+keys);
    50
    51     Iterator<Integer> itr = keys.iterator();
    52     while(itr.hasNext()) {
    53         Integer key = itr.next();
    54         String value = map.get(key);
    55
    56         System.out.println(key+" \t "+value);
    57     }
    58
    59     Set<Entry<Integer, String>> set = map.entrySet();
    60     Iterator<Entry<Integer, String>> itr1 = set.iterator();
    61
    62 }
    63
    64 }
```

8.4 Use a while loop with itr1.hasNext()

```

 31.     System.out.println(map);
 32.
 33.     String name = map.get(121);
 34.     System.out.println("name is: "+name);
 35.
 36.     if(map.containsKey(567)) {
 37.         System.out.println("567 is in the map");
 38.     }
 39.
 40.     if(map.containsValue("Dave")) {
 41.         System.out.println("Dave is in the map");
 42.     }
 43.
 44.     System.out.println("Size of map is: "+map.size());
 45.     map.remove(567);
 46.     System.out.println("Size of map after remove is: "+map.size());
 47.
 48.     Set<Integer> keys = map.keySet();
 49.     System.out.println("Keys in map are: "+keys);
 50.
 51.     Iterator<Integer> itr = keys.iterator();
 52.     while(itr.hasNext()) {
 53.         Integer key = itr.next();
 54.         String value = map.get(key);
 55.
 56.         System.out.println(key+" \t "+value);
 57.     }
 58.
 59.     Set<Entry<Integer, String>> set = map.entrySet();
 60.     Iterator<Entry<Integer, String>> itr1 = set.iterator();
 61.
 62.     while(itr1.hasNext()) {
 63.         |
 64.     }
 65.
 66. }
 67.
 68.

```

8.5 Retrieve each entry Map:

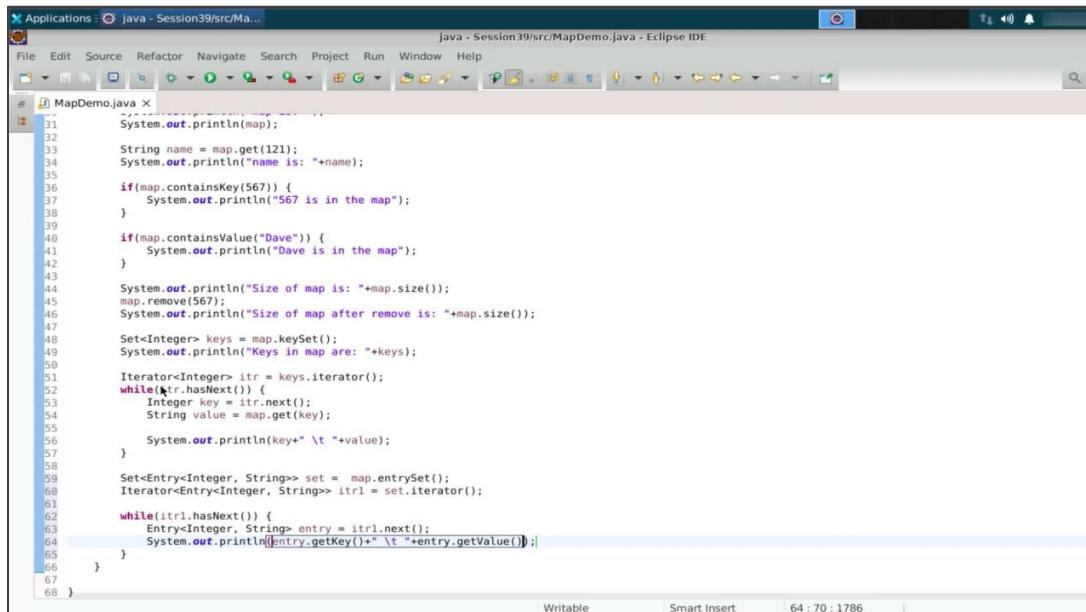
Entry<Integer, String> entry = itr1.next();

```

 31.     System.out.println(map);
 32.
 33.     String name = map.get(121);
 34.     System.out.println("name is: "+name);
 35.
 36.     if(map.containsKey(567)) {
 37.         System.out.println("567 is in the map");
 38.     }
 39.
 40.     if(map.containsValue("Dave")) {
 41.         System.out.println("Dave is in the map");
 42.     }
 43.
 44.     System.out.println("Size of map is: "+map.size());
 45.     map.remove(567);
 46.     System.out.println("Size of map after remove is: "+map.size());
 47.
 48.     Set<Integer> keys = map.keySet();
 49.     System.out.println("Key in map are: "+keys);
 50.
 51.     Iterator<Integer> itr = keys.iterator();
 52.     while(itr.hasNext()) {
 53.         Integer key = itr.next();
 54.         String value = map.get(key);
 55.
 56.         System.out.println(key+" \t "+value);
 57.     }
 58.
 59.     Set<Entry<Integer, String>> set = map.entrySet();
 60.     Iterator<Entry<Integer, String>> itr1 = set.iterator();
 61.
 62.     while(itr1.hasNext()) {
 63.         Entry<Integer, String> entry = itr1.next();
 64.     }
 65.
 66. }
 67.
 68.

```

8.6 Print the key-value pairs

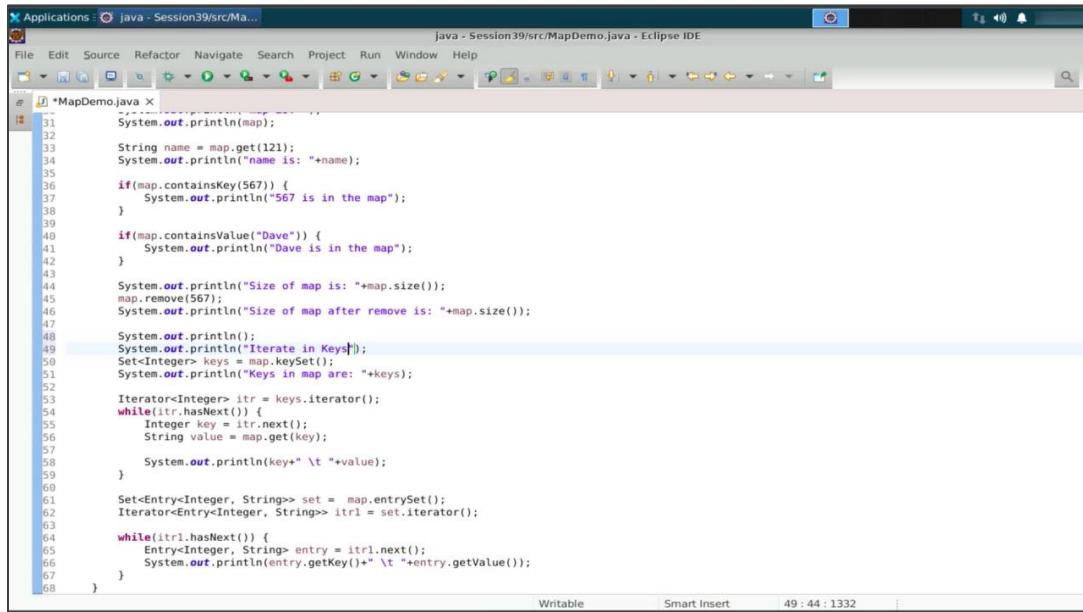


```

 31     System.out.println(map);
 32
 33     String name = map.get(121);
 34     System.out.println("name is: "+name);
 35
 36     if(map.containsKey(567)) {
 37         System.out.println("567 is in the map");
 38     }
 39
 40     if(map.containsValue("Dave")) {
 41         System.out.println("Dave is in the map");
 42     }
 43
 44     System.out.println("Size of map is: "+map.size());
 45     map.remove(567);
 46     System.out.println("Size of map after remove is: "+map.size());
 47
 48     Set<Integer> keys = map.keySet();
 49     System.out.println("Keys in map are: "+keys);
 50
 51     Iterator<Integer> itr = keys.iterator();
 52     while(itr.hasNext()) {
 53         Integer key = itr.next();
 54         String value = map.get(key);
 55
 56         System.out.println(key+" \t "+value);
 57     }
 58
 59     Set<Entry<Integer, String>> set = map.entrySet();
 60     Iterator<Entry<Integer, String>> itr1 = set.iterator();
 61
 62     while(itr1.hasNext()) {
 63         Entry<Integer, String> entry = itr1.next();
 64         System.out.println(entry.getKey()+" \t "+entry.getValue());
 65     }
 66
 67 }
 68

```

8.7 Print Iterate in Keys



```

 31     System.out.println(map);
 32
 33     String name = map.get(121);
 34     System.out.println("name is: "+name);
 35
 36     if(map.containsKey(567)) {
 37         System.out.println("567 is in the map");
 38     }
 39
 40     if(map.containsValue("Dave")) {
 41         System.out.println("Dave is in the map");
 42     }
 43
 44     System.out.println("Size of map is: "+map.size());
 45     map.remove(567);
 46     System.out.println("Size of map after remove is: "+map.size());
 47
 48     System.out.println();
 49     System.out.println("Iterate in Keys");
 50     Set<Integer> keys = map.keySet();
 51     System.out.println("Keys in map are: "+keys);
 52
 53     Iterator<Integer> itr = keys.iterator();
 54     while(itr.hasNext()) {
 55         Integer key = itr.next();
 56         String value = map.get(key);
 57
 58         System.out.println(key+" \t "+value);
 59     }
 60
 61     Set<Entry<Integer, String>> set = map.entrySet();
 62     Iterator<Entry<Integer, String>> itr1 = set.iterator();
 63
 64     while(itr1.hasNext()) {
 65         Entry<Integer, String> entry = itr1.next();
 66         System.out.println(entry.getKey()+" \t "+entry.getValue());
 67     }
 68

```

8.8 Print Iterate using entry set

```

 1 *MapDemo.java X
 2
 3 import java.util.*;
 4
 5 public class MapDemo {
 6     public static void main(String[] args) {
 7         Map<Integer, String> map = new HashMap<Integer, String>();
 8         map.put(101, "John");
 9         map.put(121, "Dave");
10         map.put(322, "Anna");
11         map.put(334, "Mike");
12         map.put(777, null);
13         map.put(888, null);
14         map.put(891, "Anna");
15
16         System.out.println("Map is: " + map);
17         System.out.println("Name is: " + name);
18
19         if(map.containsKey(567)) {
20             System.out.println("567 is in the map");
21         }
22
23         if(map.containsValue("Dave")) {
24             System.out.println("Dave is in the map");
25         }
26
27         System.out.println("Size of map is: " + map.size());
28         map.remove(567);
29         System.out.println("Size of map after remove is: " + map.size());
30
31         System.out.println();
32         System.out.println("Iterate in Keys");
33         Set<Integer> keys = map.keySet();
34         System.out.println("Keys in map are: " + keys);
35
36         Iterator<Integer> itr = keys.iterator();
37         while(itr.hasNext()) {
38             Integer key = itr.next();
39             String value = map.get(key);
40
41             System.out.println(key+ " : " + value);
42         }
43
44         System.out.println();
45         System.out.println("Iterate using entry set");
46
47         Set<Entry<Integer, String>> set = map.entrySet();
48         Iterator<Entry<Integer, String>> itrl = set.iterator();
49
50         while(itrl.hasNext()) {
51             Entry<Integer, String> entry = itrl.next();
52
53             System.out.println(entry.getKey() + " : " + entry.getValue());
54         }
55
56         System.out.println();
57         System.out.println("Iterate using entry set");
58
59         Set<Entry<Integer, String>> set = map.entrySet();
60         Iterator<Entry<Integer, String>> itrl = set.iterator();
61
62         while(itrl.hasNext()) {
63             Entry<Integer, String> entry = itrl.next();
64
65             System.out.println(entry.getKey() + " : " + entry.getValue());
66         }
67
68     }
69 }

```

8.9 Run the code and observe the printed key-value pairs

The screenshot shows the Eclipse IDE interface with the Java code for MapDemo.java. The code is identical to the one in the previous screenshot. To the right of the editor, the Eclipse IDE's integrated terminal window (Console view) is open, showing the output of the application's execution. The output consists of two parts: the initial state of the map and the results of two iteration loops.

Map State:

```

<terminated> MapDemo [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core
Keys in map are: [101, 121, 322, 334, 777, 888, 891]
101 John
121 Dave
322 Anna
334 Mike
777 null
888 null
891 Anna

```

Iteration Output:

```

Iterate using entry set
101 John
121 Dave
322 Anna
334 Mike
777 null
888 null
891 Anna

```

Step 9: Work with a Hashtable

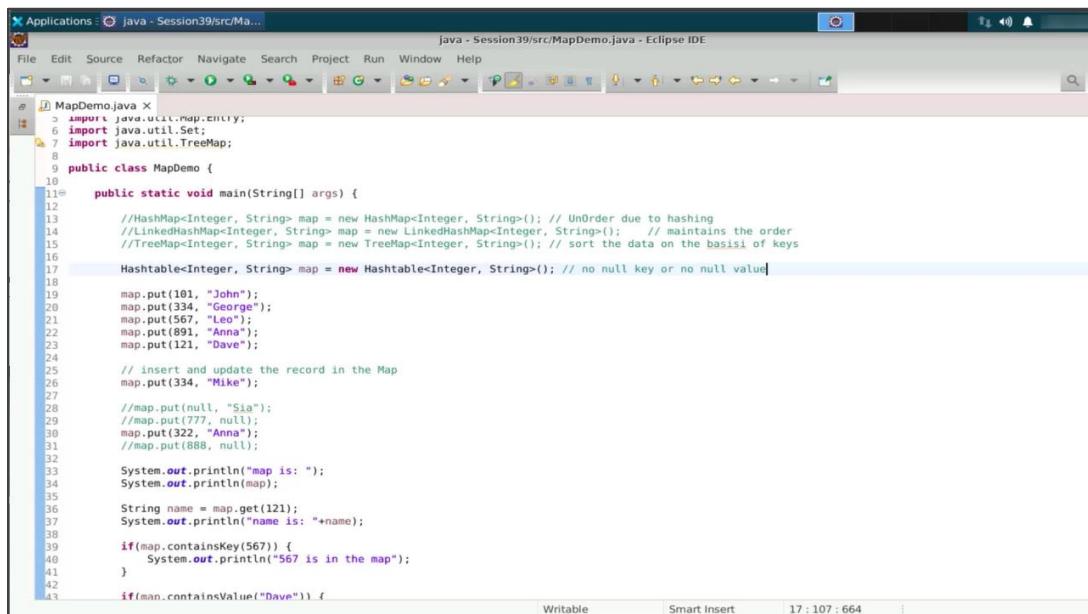
9.1 Create a Hashtable named **map**

The screenshot shows the Eclipse IDE interface with the title bar "Applications - java - Session39/src/Ma...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, etc. The left sidebar shows the project structure with "MapDemo.java" selected. The main editor area contains the following Java code:

```
1 import java.util.HashMap;
2 import java.util.Hashtable;
3 import java.util.Iterator;
4 import java.util.LinkedHashMap;
5 import java.util.Map.Entry;
6 import java.util.Set;
7 import java.util.TreeMap;
8
9 public class MapDemo {
10
11    public static void main(String[] args) {
12
13        //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
14        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
15        //TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
16
17        Hashtable<Integer, String> map = new Hashtable<Integer, String>();
18
19        map.put(101, "John");
20        map.put(334, "George");
21        map.put(567, "Leo");
22        map.put(891, "Anna");
23        map.put(121, "Dave");
24
25        // insert and update the record in the Map
26        map.put(334, "Mike");
27
28        //map.put(null, "Sia");
29        map.put(777, null);
30        map.put(322, "Anna");
31        map.put(888, null);
32
33        System.out.println("map is: ");
34        System.out.println(map);
35
36        String name = map.get(121);
37        System.out.println("name is: "+name);
38
```

Note: Hashtable does not allow null keys or values.

9.2 Add a comment indicating the restriction on null keys or values



The screenshot shows the Eclipse IDE interface with the title bar "Applications : Java - Session39/src/MapDemo.java" and "File - Session39/src/MapDemo.java - Eclipse IDE". The code editor displays the following Java code:

```
MapDemo.java X
1 import java.util.Map;
2 import java.util.Set;
3 import java.util.TreeMap;
4
5 public class MapDemo {
6
7     public static void main(String[] args) {
8
9         //HashMap<Integer, String> map = new HashMap<Integer, String>(); // UnOrder due to hashing
10        //LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>(); // maintains the order
11        //TreeMap<Integer, String> map = new TreeMap<Integer, String>(); // sort the data on the basis of keys
12
13        Hashtable<Integer, String> map = new Hashtable<Integer, String>(); // no null key or no null value
14
15        map.put(101, "John");
16        map.put(334, "George");
17        map.put(567, "Leo");
18        map.put(891, "Anna");
19        map.put(121, "Dave");
20
21        // insert and update the record in the Map
22        map.put(334, "Mike");
23
24        //map.put(null, "Sia");
25        //map.put(777, null);
26        map.put(322, "Anna");
27        //map.put(888, null);
28
29        System.out.println("map is: ");
30        System.out.println(map);
31
32        String name = map.get(121);
33        System.out.println("name is: "+name);
34
35        if(map.containsKey(567)) {
36            System.out.println("567 is in the map");
37        }
38
39        if(map.containsValue("Dave")) {
40            System.out.println("Dave is in the map");
41        }
42
43    }
44}
```

By following these steps, you have successfully implemented and demonstrated various functionalities of HashMap, LinkedHashMap, and Hashtable in Java.