Container Classes

Maps, Sorted Maps

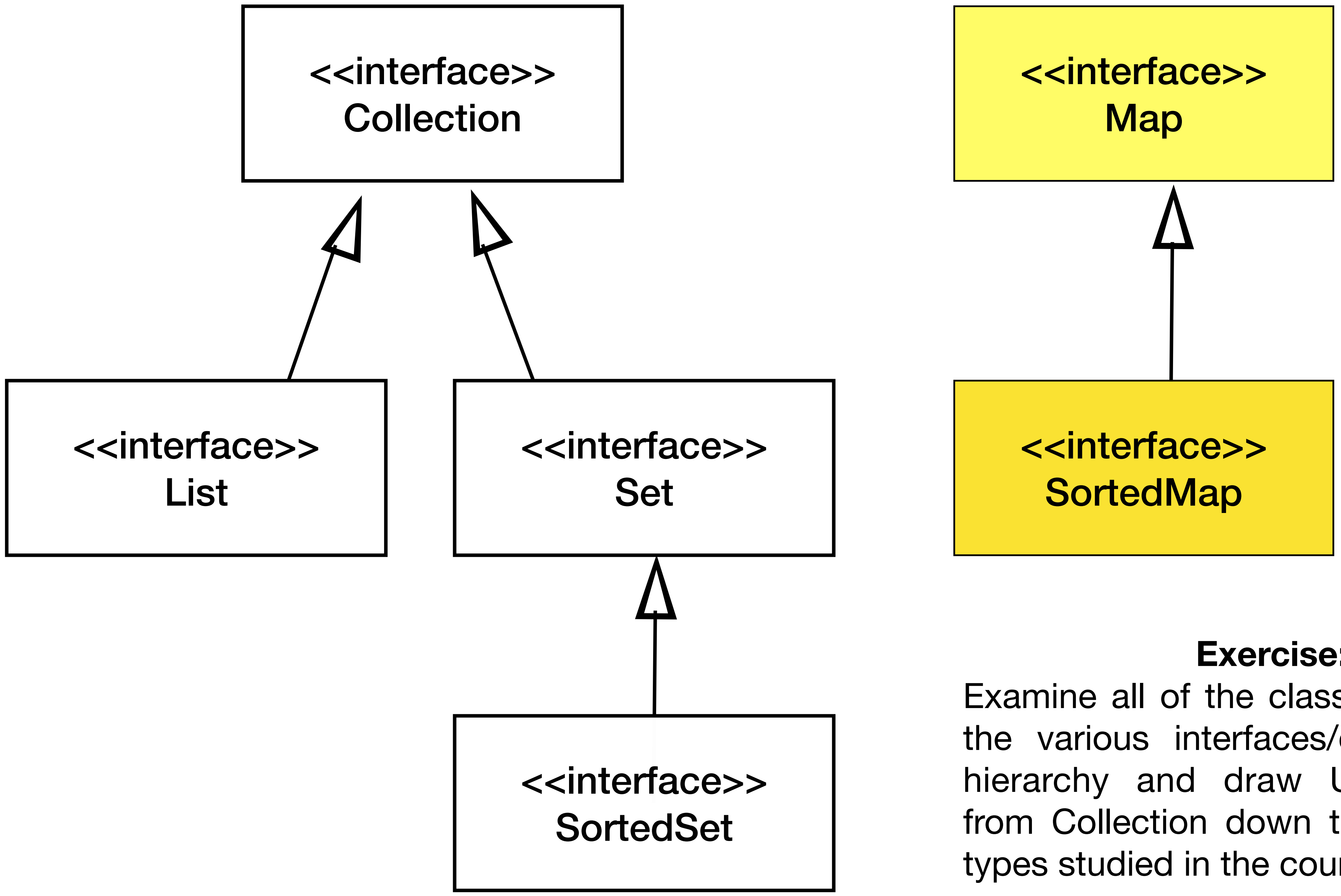
COMP2603
Object Oriented Programming 1

Week 9, Lecture 1

Outline

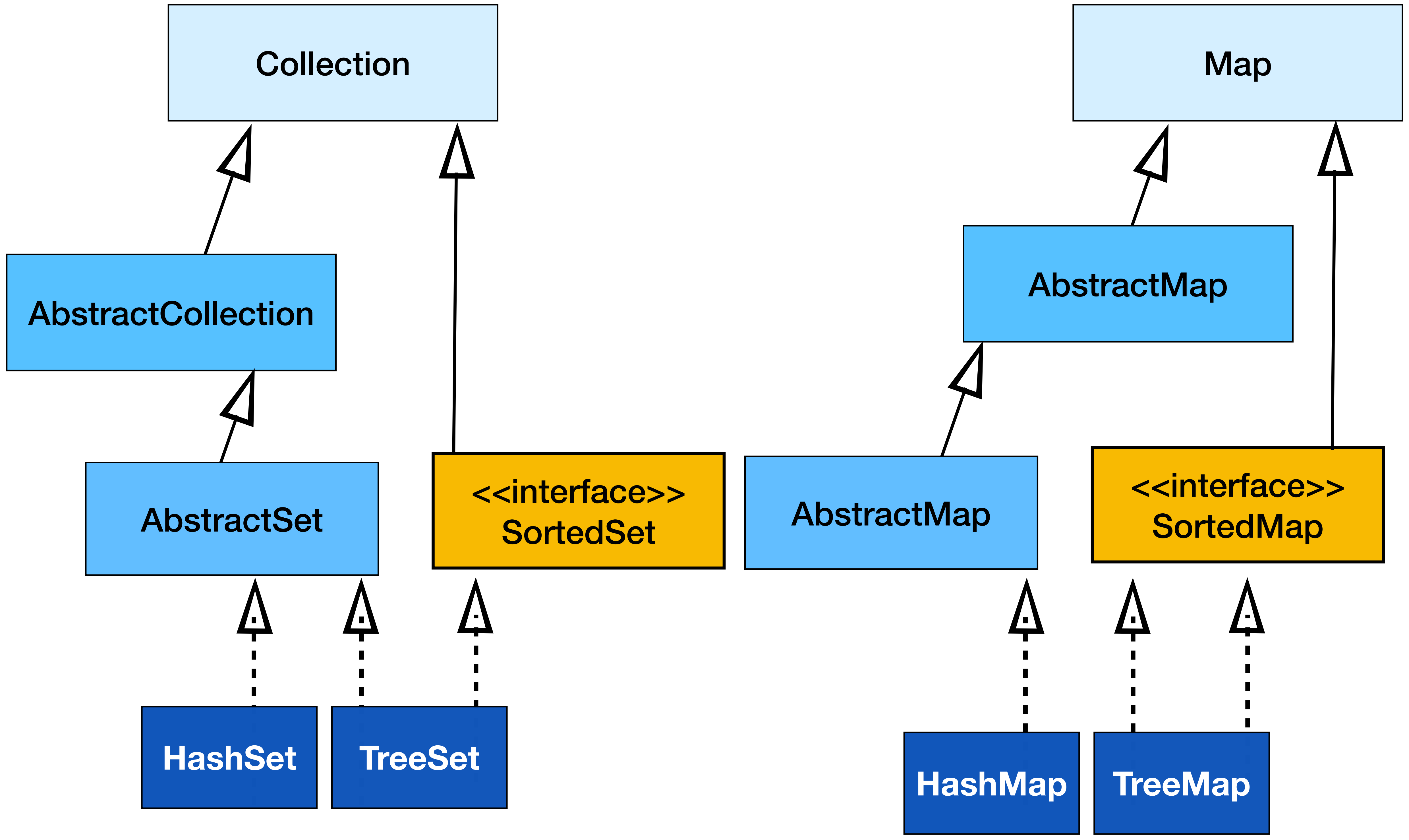
- Java Map Interface
    - Map
        - HashMap
        - TreeMap

Interfaces in the Java Collections Framework



Exercise:
Examine all of the class signatures for the various interfaces/classes in the hierarchy and draw UML diagrams from Collection down to the concrete types studied in the course.

Classes in the Java Collections Framework



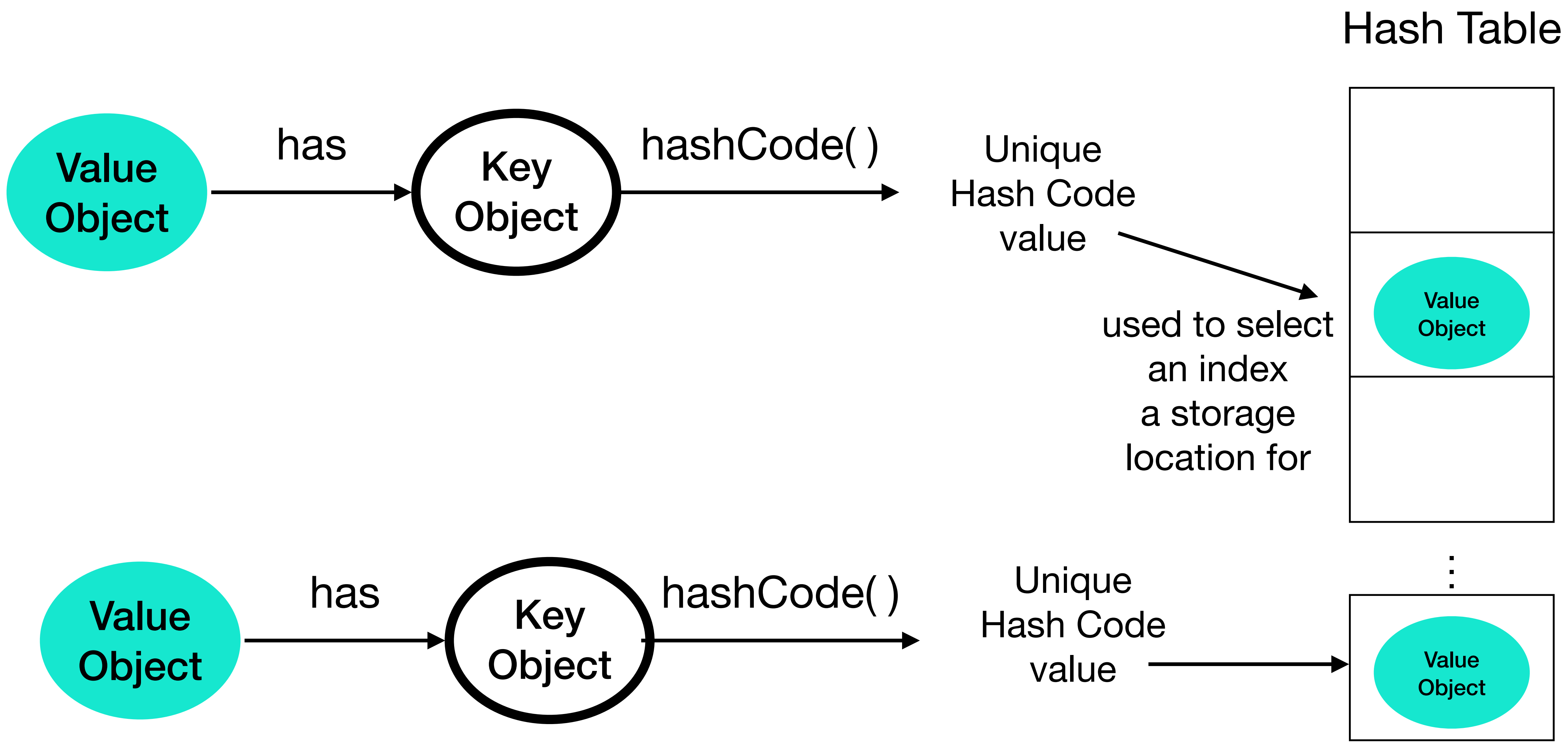
The Map Interface

The Map interface represents a collection of mappings between **key** objects and **value** objects.

Hash tables are examples of maps.

The set of key objects in a Map must not have any duplicates. However, the collection of value objects may contain duplicates.

Maps


Hash Table
Value Object
has
Key Object
hashCode( )
Unique Hash Code value
used to select an index a storage location for
Value Object
Value Object
has
Key Object
hashCode( )
Unique Hash Code value
Value Object

The Map Interface

Method	Description
boolean containsKey(Object key)	Returns true if the Map contains a mapping for the specified key, and false otherwise
V get (Object key)	Returns the value object associated with the specified key or null if there is no mapping for the key
Set<E> keySet( )	Returns a Set of all the key objects in the Map
V put(K key, V value)	Creates a key/value mapping in the Map. If the key already exists in the Map, put( ) replaces the value currently in the Map with the value supplied as an argument and returns the value replaced; otherwise it returns the value.
Collection<V> values( )	Returns a Collection of all the value objects in the Map

https://docs.oracle.com/javase/7/docs/api/java/util/Map.html

HashMap

Hash table based implementation of the Map interface.

This implementation provides all of the optional map operations, and permits null values and the null key.

(The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.)

This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time

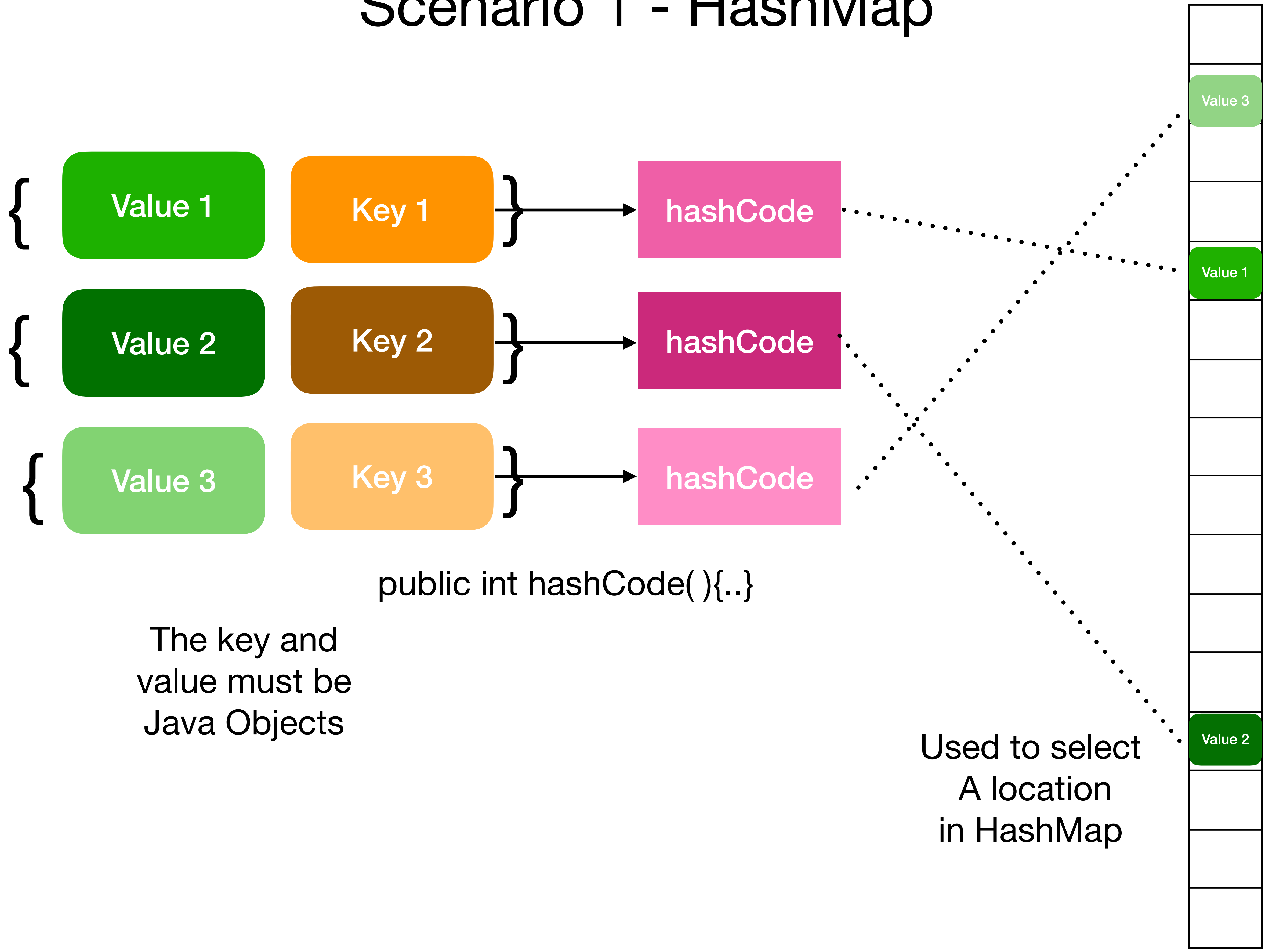
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/HashMap.html

The Map Interface
Preventing duplicate keys

To ensure that the set of key objects in the Map does not contain duplicates, it is important that the Key objects override the **equals( )** method of the Object class, based on the content of the key.

Duplicates depend on how the equals( ) method is defined.

Scenario 1 - HashMap

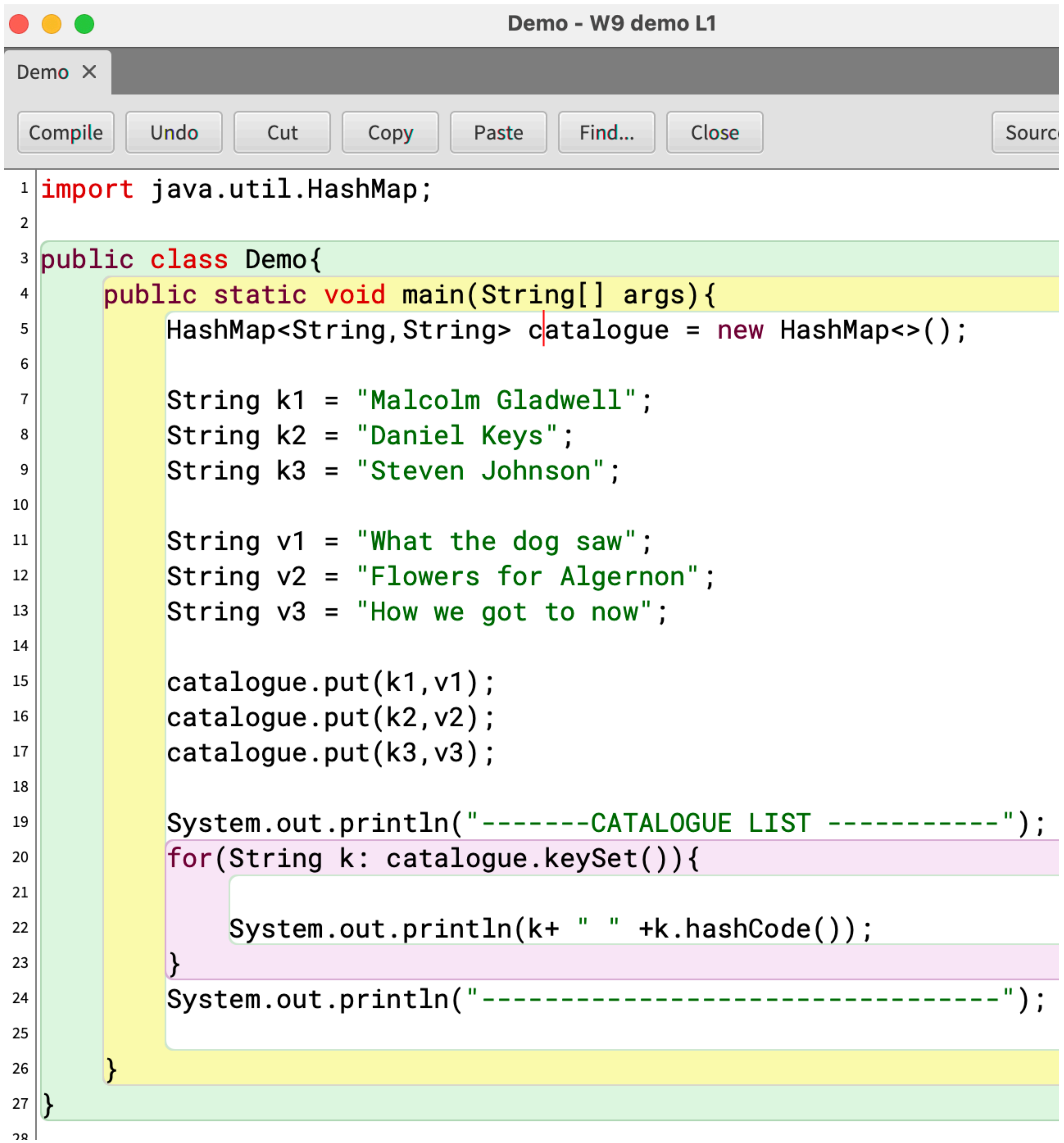


public int hashCode( ){..}

The key and value must be Java Objects

Used to select A location in HashMap

Example 1 - HashMap



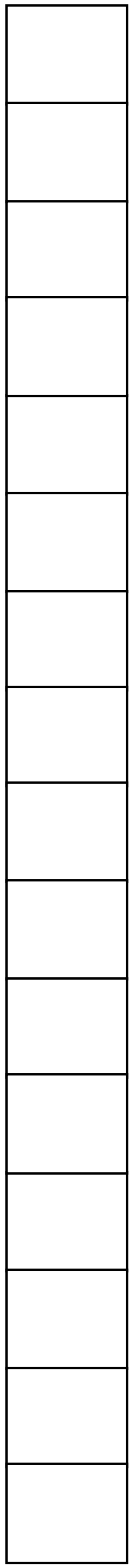
```java
import java.util.HashMap;

public class Demo{
    public static void main(String[] args){
        HashMap<String,String> catalogue = new HashMap<>();

        String k1 = "Malcolm Gladwell";
        String k2 = "Daniel Keys";
        String k3 = "Steven Johnson";

        String v1 = "What the dog saw";
        String v2 = "Flowers for Algernon";
        String v3 = "How we got to now";

        catalogue.put(k1,v1);
        catalogue.put(k2,v2);
        catalogue.put(k3,v3);

        System.out.println("-------CATALOGUE LIST -----------");
        for(String k: catalogue.keySet()){

            System.out.println(k+ " " +k.hashCode());
        }
        System.out.println("---------------------------------");

    }
}
```

{key,
value}

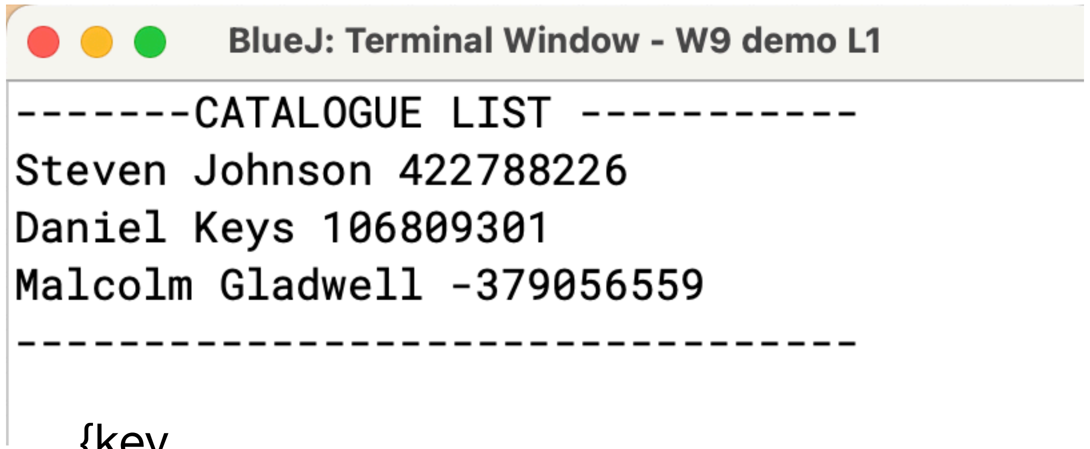
{“Malcolm Gladwell”,
“What the dog saw”}

{“Daniel Keys”,
“Flowers for Algernon”}

{“Steven Johnson”,
“How we got to now”}

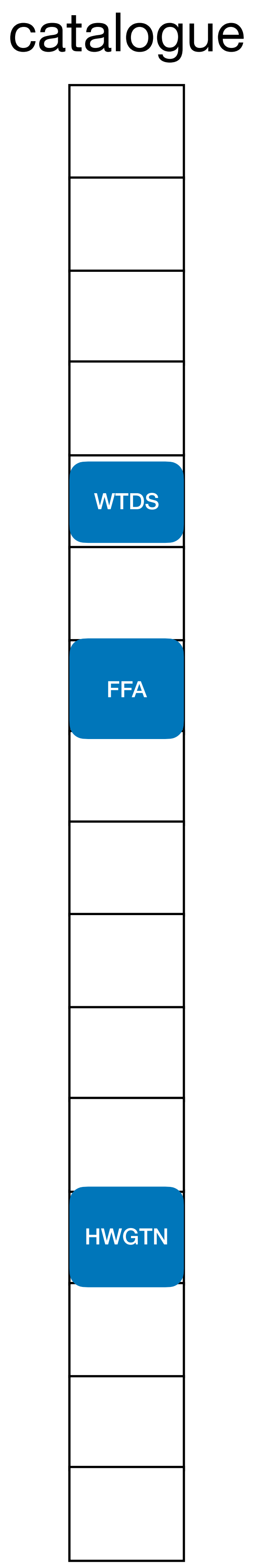


Example 1 - HashMap



```
-------CATALOGUE LIST -----------
Steven Johnson 422788226
Daniel Keys 106809301
Malcolm Gladwell -379056559
---------------------------------
```

{key, value}	
{"Malcolm Gladwell", "What the dog saw"}	-379056559
{"Daniel Keys", "Flowers for Algernon"}	106809301
{"Steven Johnson", "How we got to now"}	422788226



String hashCode( )

“In the String class, hashCode is computed by the following formula

s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)

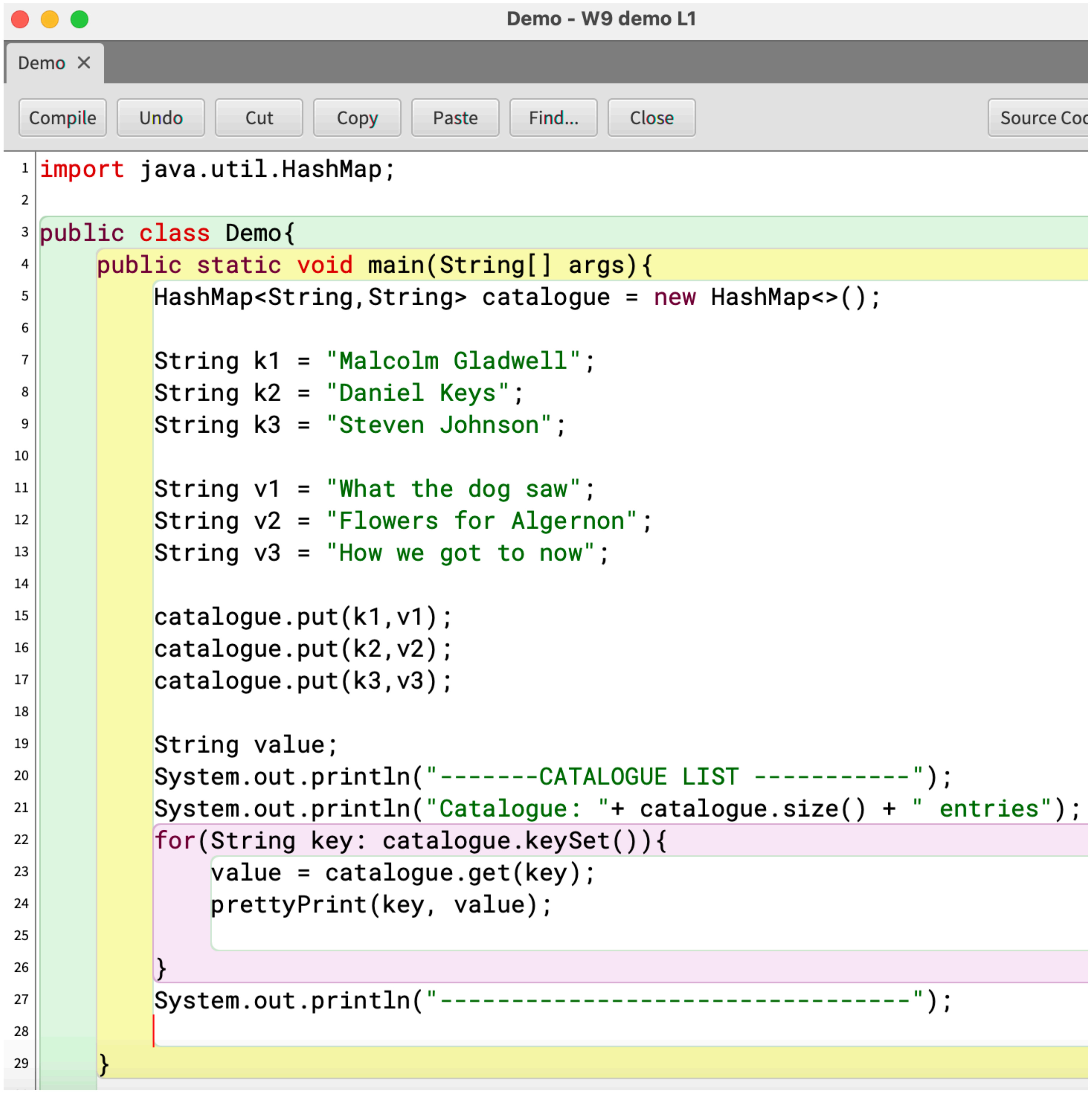
where s is a string and n is its length. An example

"ABC" = 'A' * 31^2 + 'B' * 31 + 'C' = 65 * 31^2 + 66 * 31 + 67 = 64578

Note that Java's hashCode method might return a negative integer. If a string is long enough, its hashcode will be bigger than the largest integer we can store on 32 bits CPU. In this case, due to integer overflow, the value returned by hashCode can be negative.”

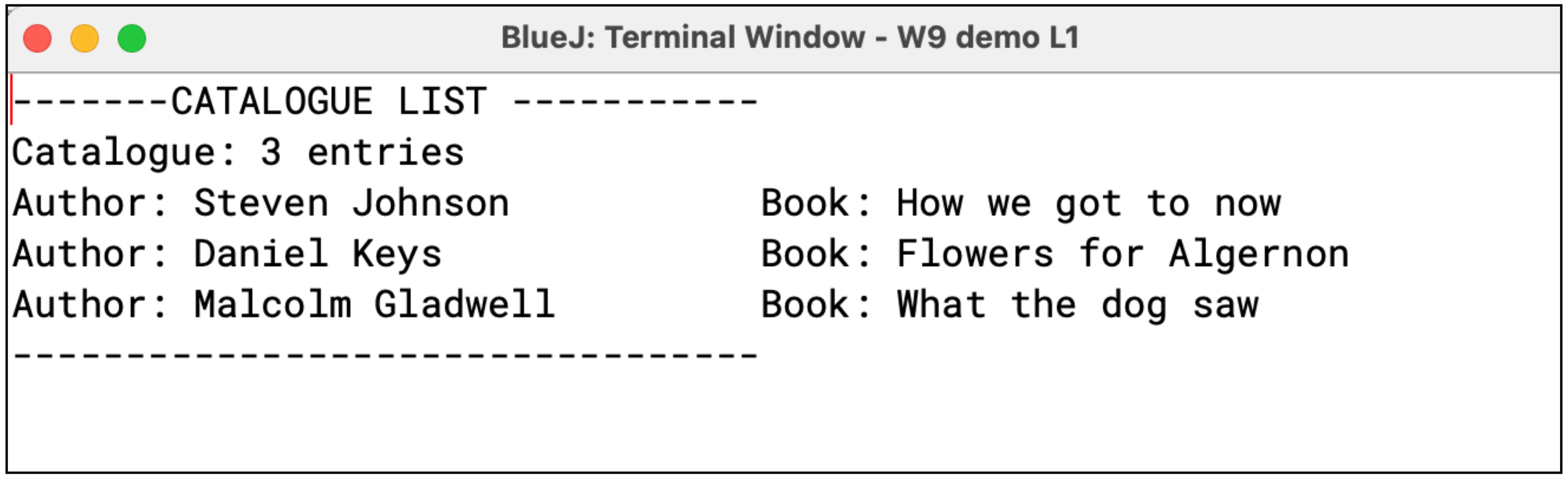
Source: https://viterbi-web.usc.edu/~adamchik/15-121/lectures/Hashing/hashing.html

Example 1 - HashMap

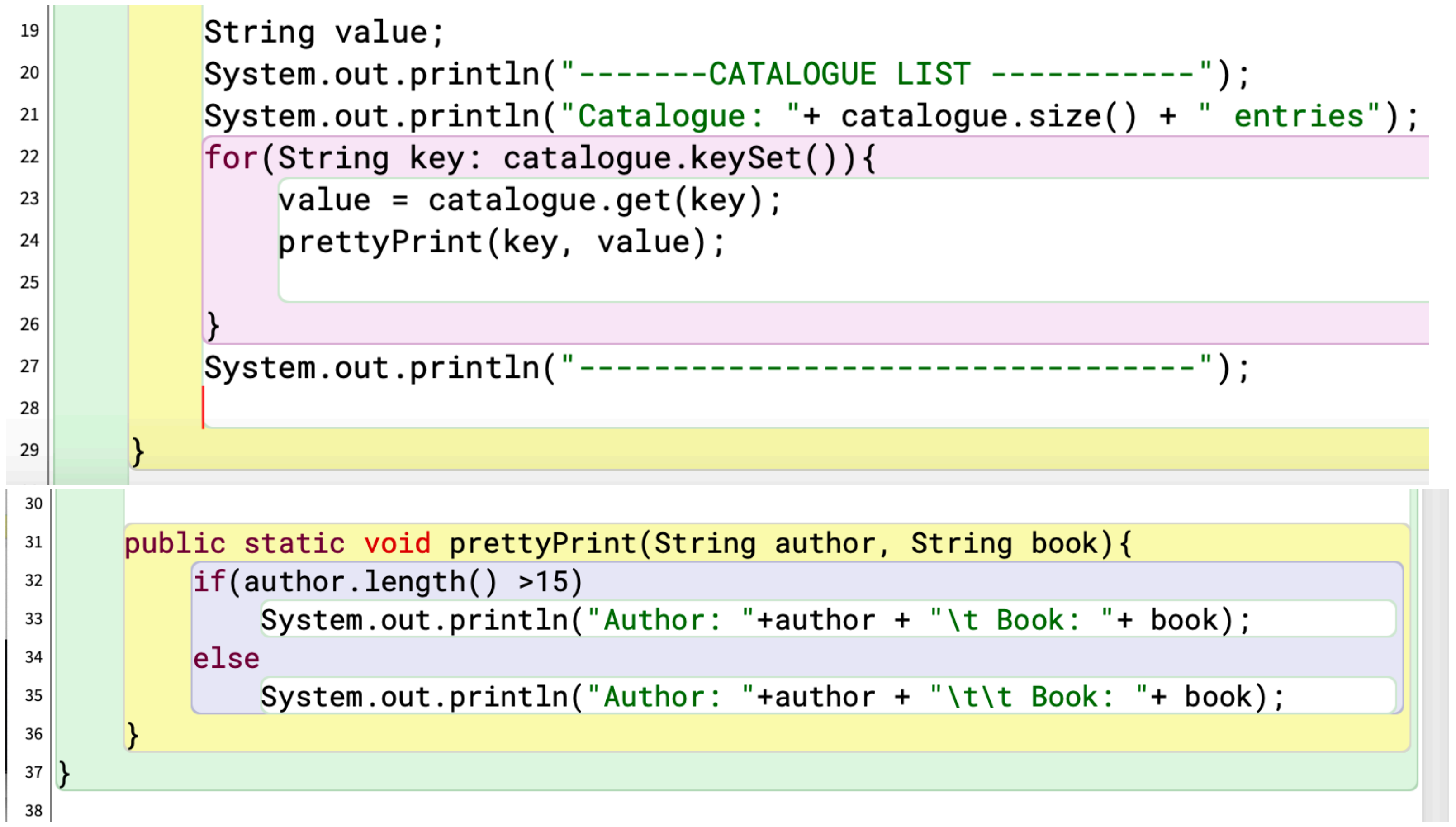


```java
import java.util.HashMap;

public class Demo{
    public static void main(String[] args){
        HashMap<String,String> catalogue = new HashMap<>();

        String k1 = "Malcolm Gladwell";
        String k2 = "Daniel Keys";
        String k3 = "Steven Johnson";

        String v1 = "What the dog saw";
        String v2 = "Flowers for Algernon";
        String v3 = "How we got to now";

        catalogue.put(k1,v1);
        catalogue.put(k2,v2);
        catalogue.put(k3,v3);

        String value;
        System.out.println("-------CATALOGUE LIST -----------");
        System.out.println("Catalogue: "+ catalogue.size() + " entries");
        for(String key: catalogue.keySet()){
            value = catalogue.get(key);
            prettyPrint(key, value);

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

    }
```

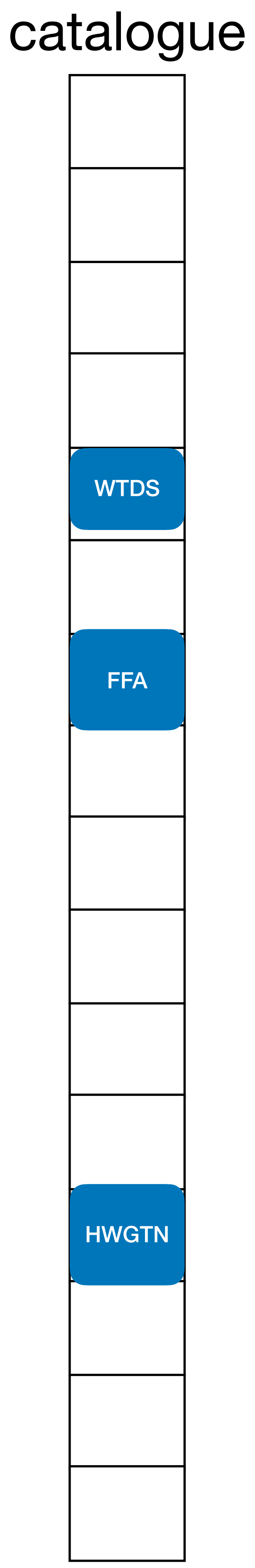
Example 1 - HashMap



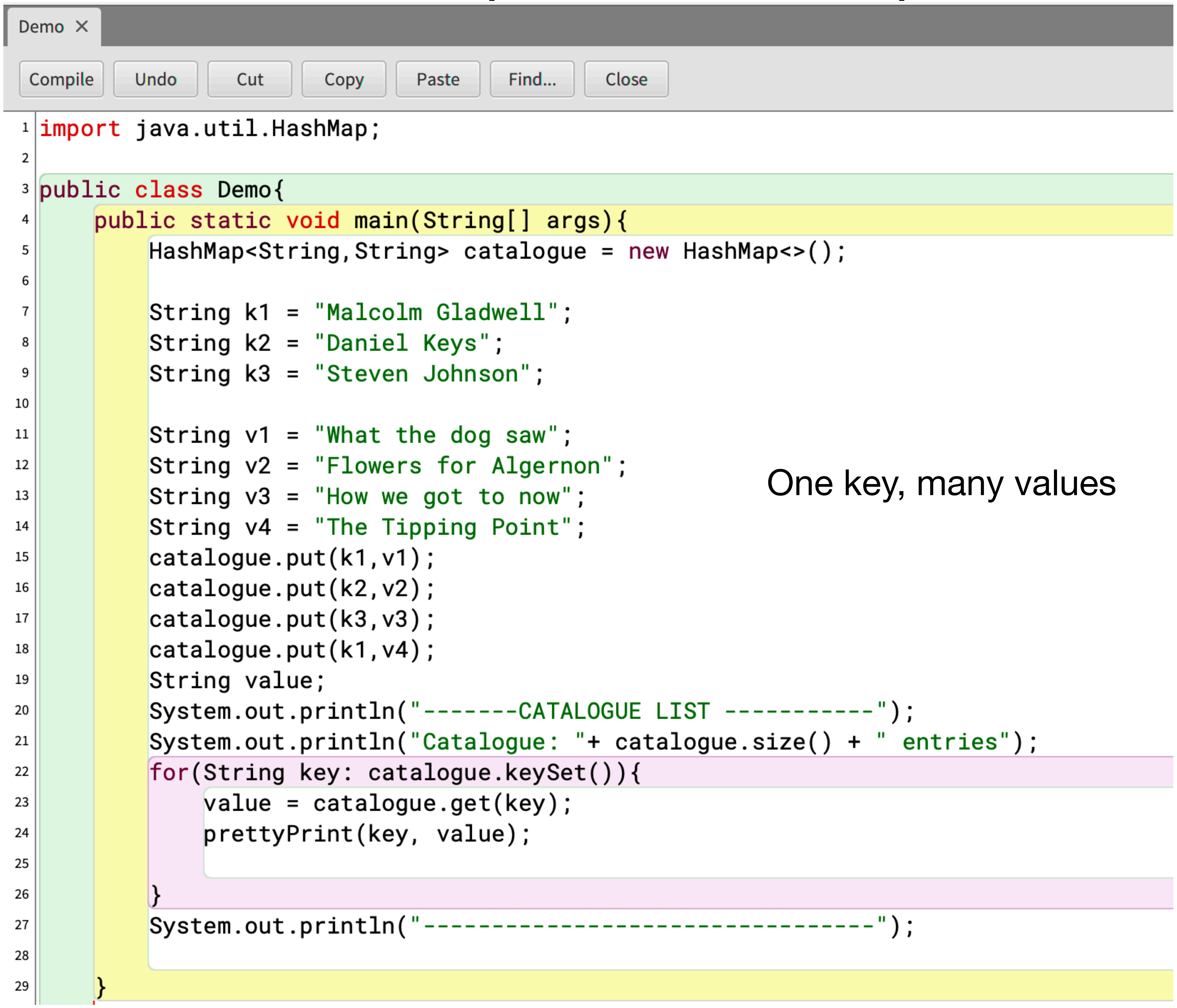
```
-------CATALOGUE LIST -----------
Catalogue: 3 entries
Author: Steven Johnson          Book: How we got to now
Author: Daniel Keys             Book: Flowers for Algernon
Author: Malcolm Gladwell        Book: What the dog saw
--------------------------------
```



```java
        String value;
        System.out.println("-------CATALOGUE LIST -----------");
        System.out.println("Catalogue: "+ catalogue.size() + " entries");
        for(String key: catalogue.keySet()){
            value = catalogue.get(key);
            prettyPrint(key, value);

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

    }

    public static void prettyPrint(String author, String book){
        if(author.length() >15)
            System.out.println("Author: "+author + "\t Book: "+ book);
        else
            System.out.println("Author: "+author + "\t\t Book: "+ book);
    }
}
```



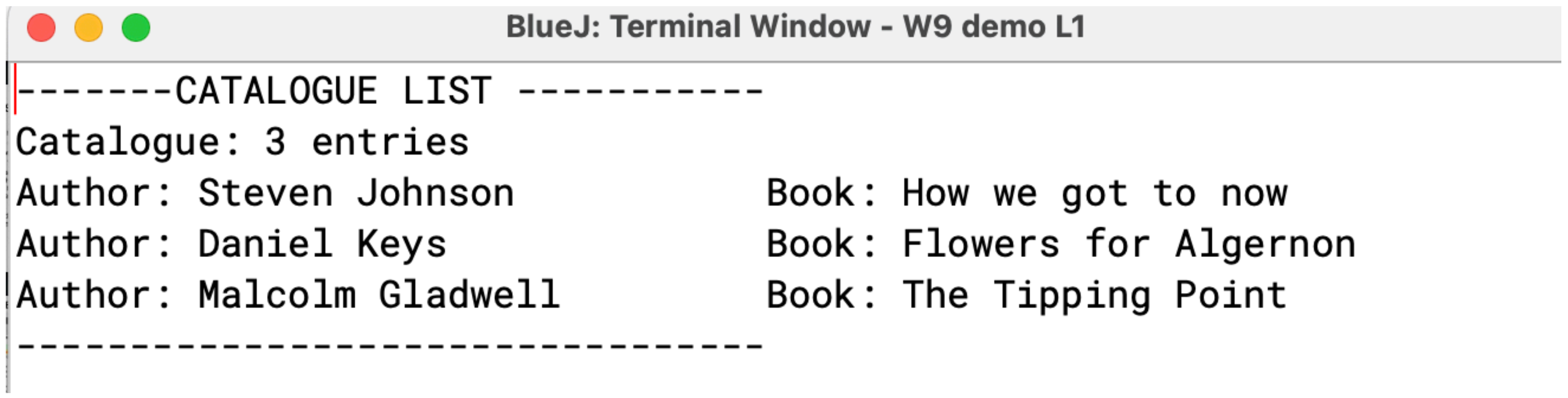
Example 2 - HashMap



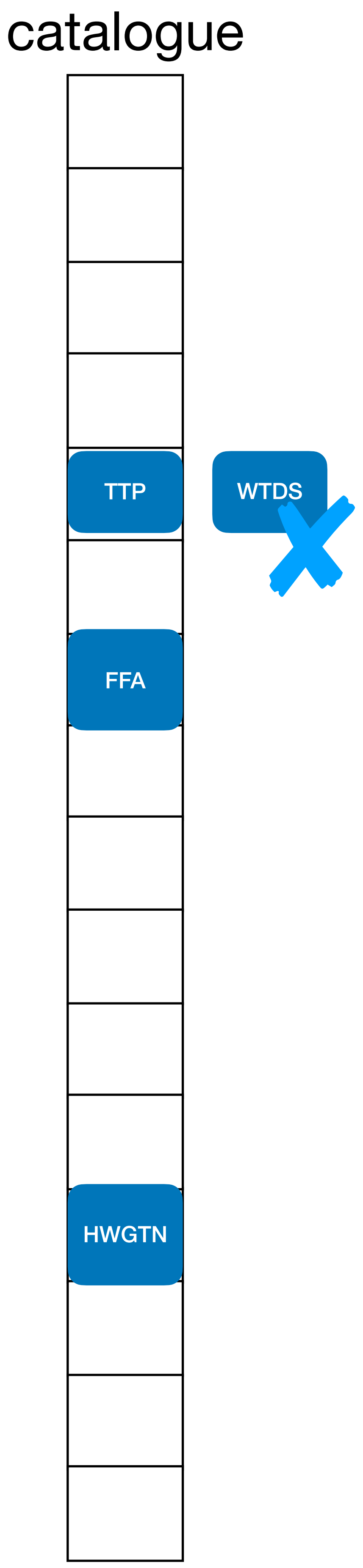
```java
import java.util.HashMap;

public class Demo{
    public static void main(String[] args){
        HashMap<String,String> catalogue = new HashMap<>();

        String k1 = "Malcolm Gladwell";
        String k2 = "Daniel Keys";
        String k3 = "Steven Johnson";

        String v1 = "What the dog saw";
        String v2 = "Flowers for Algernon";
        String v3 = "How we got to now";
        String v4 = "The Tipping Point";
        catalogue.put(k1,v1);
        catalogue.put(k2,v2);
        catalogue.put(k3,v3);
        catalogue.put(k1,v4);
        String value;
        System.out.println("-------CATALOGUE LIST -----------");
        System.out.println("Catalogue: "+ catalogue.size() + " entries");
        for(String key: catalogue.keySet()){
            value = catalogue.get(key);
            prettyPrint(key, value);

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

    }
```

One key, many values

Example 2 - HashMap

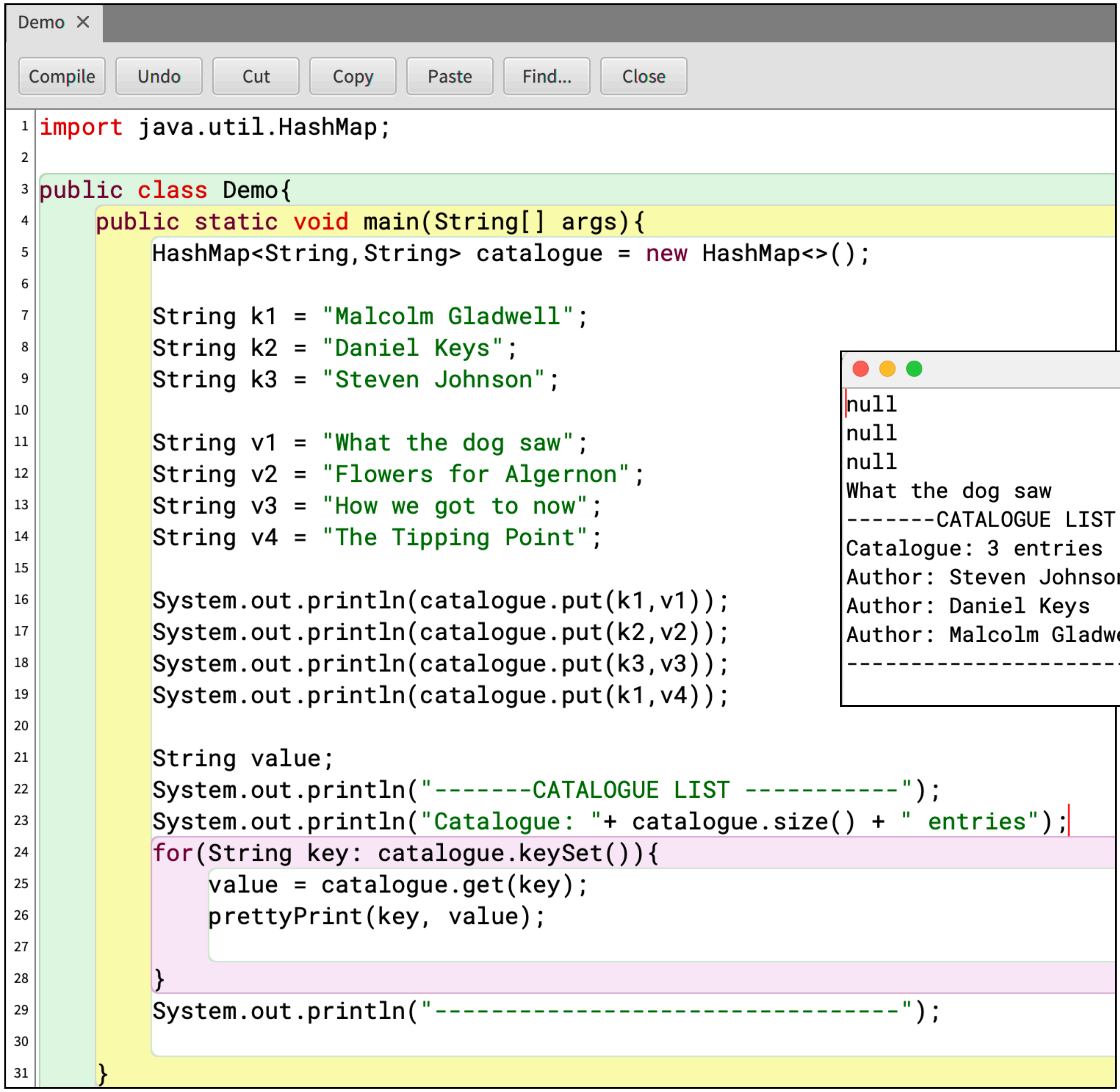


```
-------CATALOGUE LIST -----------
Catalogue: 3 entries
Author: Steven Johnson          Book: How we got to now
Author: Daniel Keys             Book: Flowers for Algernon
Author: Malcolm Gladwell        Book: The Tipping Point
---------------------------------
```

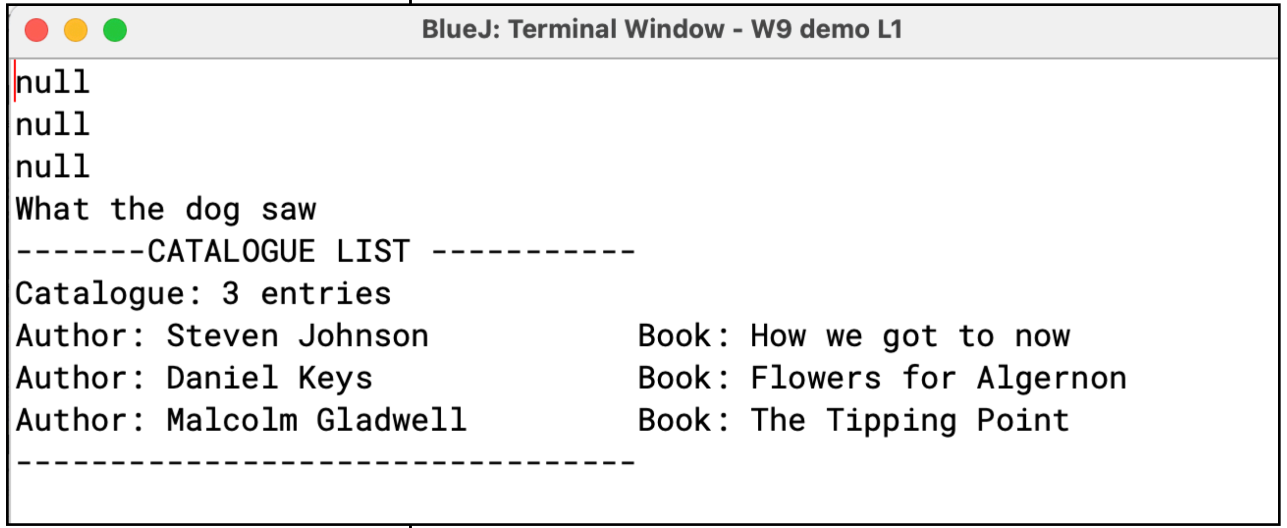


k3 already exists as a key in the catalogue HashMap. When the 4th book (which has k3 as its key) is inserted into the catalogue, the existing value is replaced

Example 2 - HashMap



```java
import java.util.HashMap;

public class Demo{
    public static void main(String[] args){
        HashMap<String,String> catalogue = new HashMap<>();

        String k1 = "Malcolm Gladwell";
        String k2 = "Daniel Keys";
        String k3 = "Steven Johnson";

        String v1 = "What the dog saw";
        String v2 = "Flowers for Algernon";
        String v3 = "How we got to now";
        String v4 = "The Tipping Point";

        System.out.println(catalogue.put(k1,v1));
        System.out.println(catalogue.put(k2,v2));
        System.out.println(catalogue.put(k3,v3));
        System.out.println(catalogue.put(k1,v4));

        String value;
        System.out.println("-------CATALOGUE LIST -----------");
        System.out.println("Catalogue: "+ catalogue.size() + " entries");
        for(String key: catalogue.keySet()){
            value = catalogue.get(key);
            prettyPrint(key, value);

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

    }
```

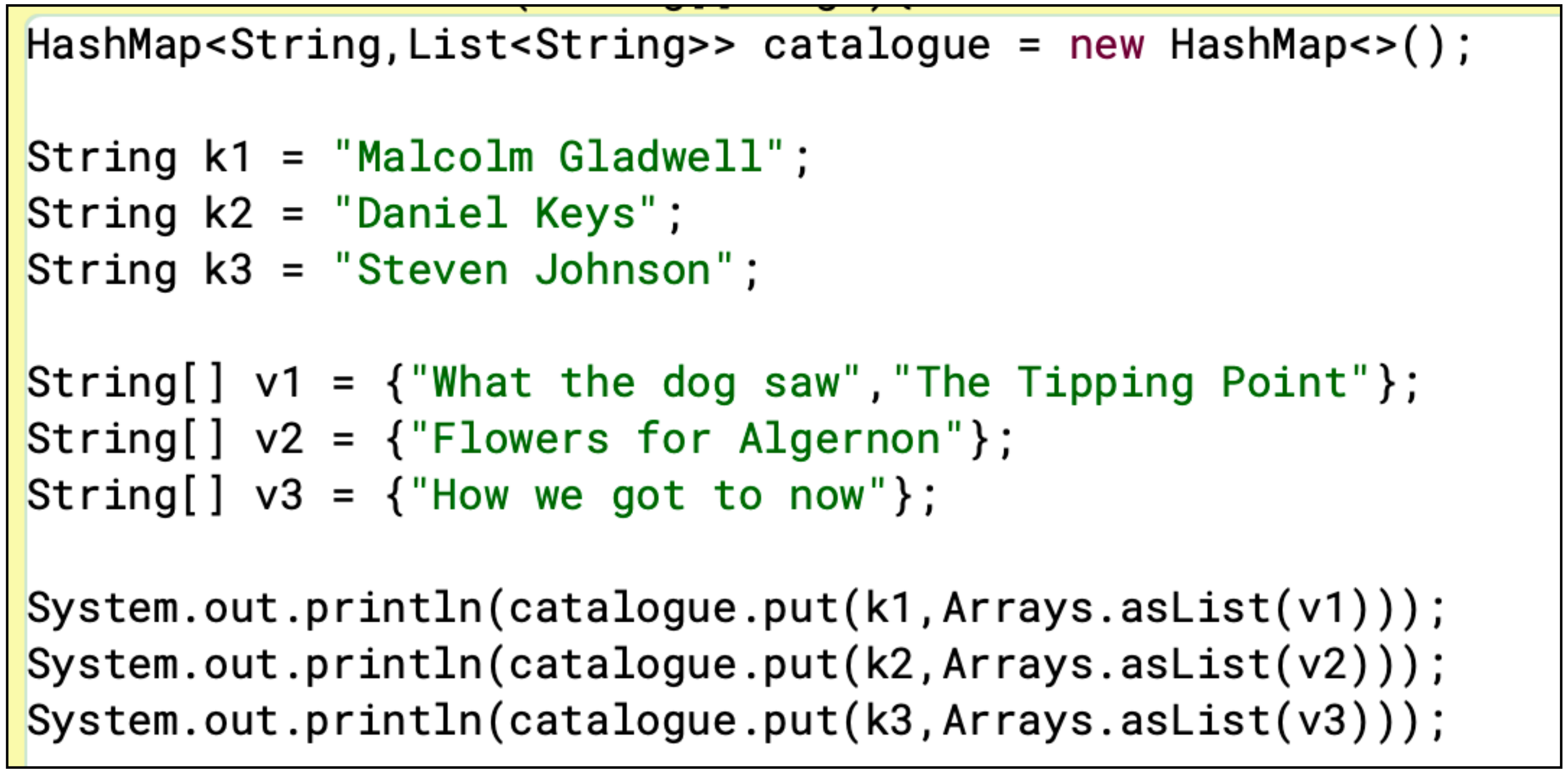


```
null
null
null
What the dog saw
-------CATALOGUE LIST -----------
Catalogue: 3 entries
Author: Steven Johnson          Book: How we got to now
Author: Daniel Keys             Book: Flowers for Algernon
Author: Malcolm Gladwell        Book: The Tipping Point
--------------------------------
```

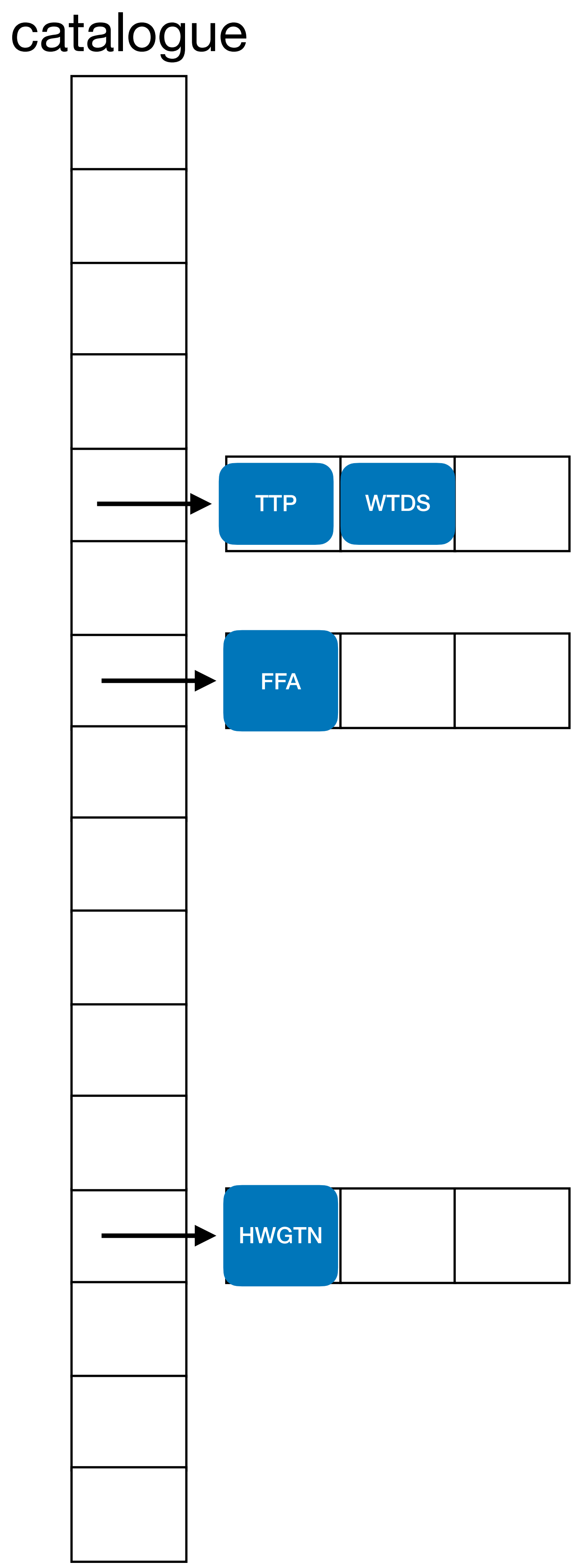
The put(..) method returns the object that was replaced (if any) in the HashMap

Example 2 - HashMap

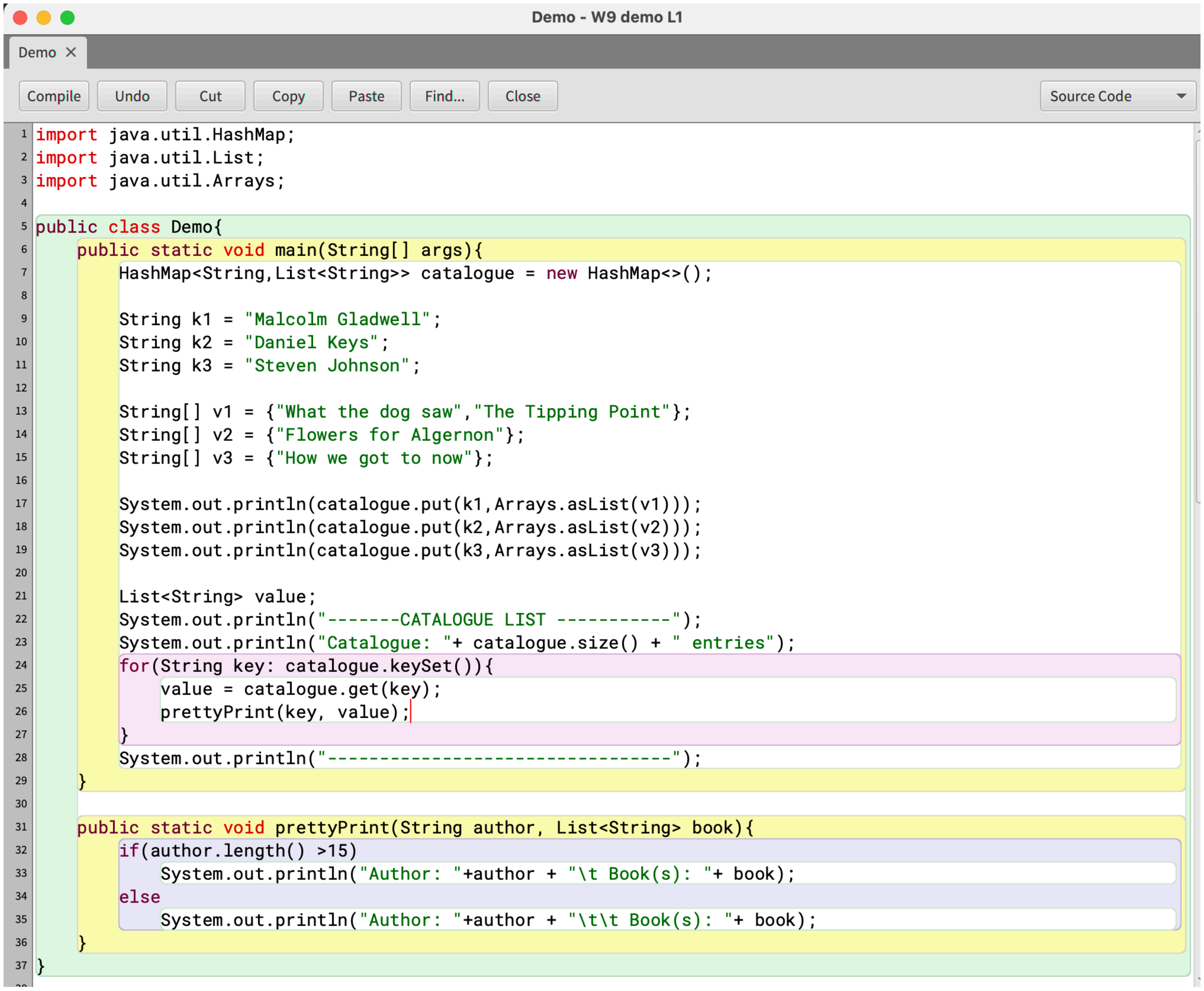
When there are multiple values associated with a key, a nested collection is often used



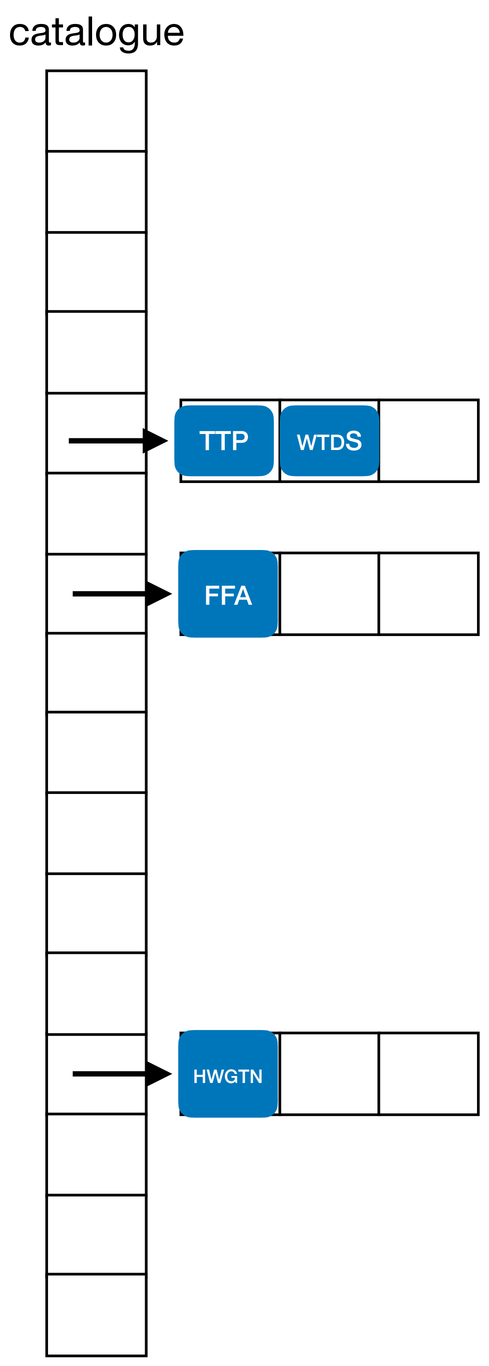
```java
HashMap<String,List<String>> catalogue = new HashMap<>();

String k1 = "Malcolm Gladwell";
String k2 = "Daniel Keys";
String k3 = "Steven Johnson";

String[] v1 = {"What the dog saw","The Tipping Point"};
String[] v2 = {"Flowers for Algernon"};
String[] v3 = {"How we got to now"};

System.out.println(catalogue.put(k1,Arrays.asList(v1)));
System.out.println(catalogue.put(k2,Arrays.asList(v2)));
System.out.println(catalogue.put(k3,Arrays.asList(v3)));
```



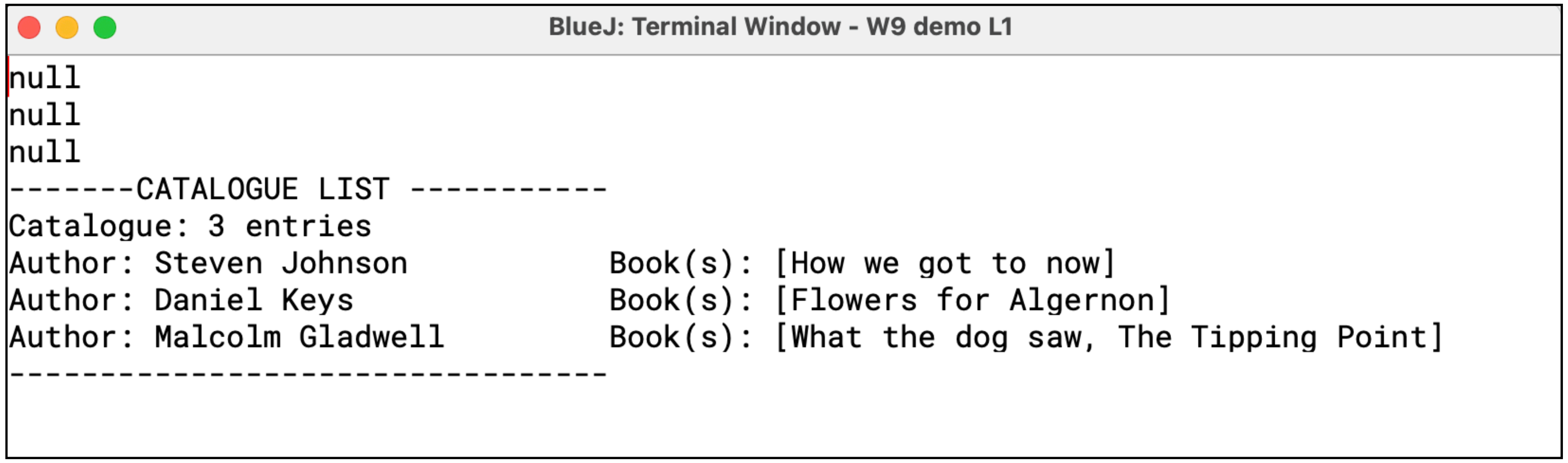
Example 2 - HashMap



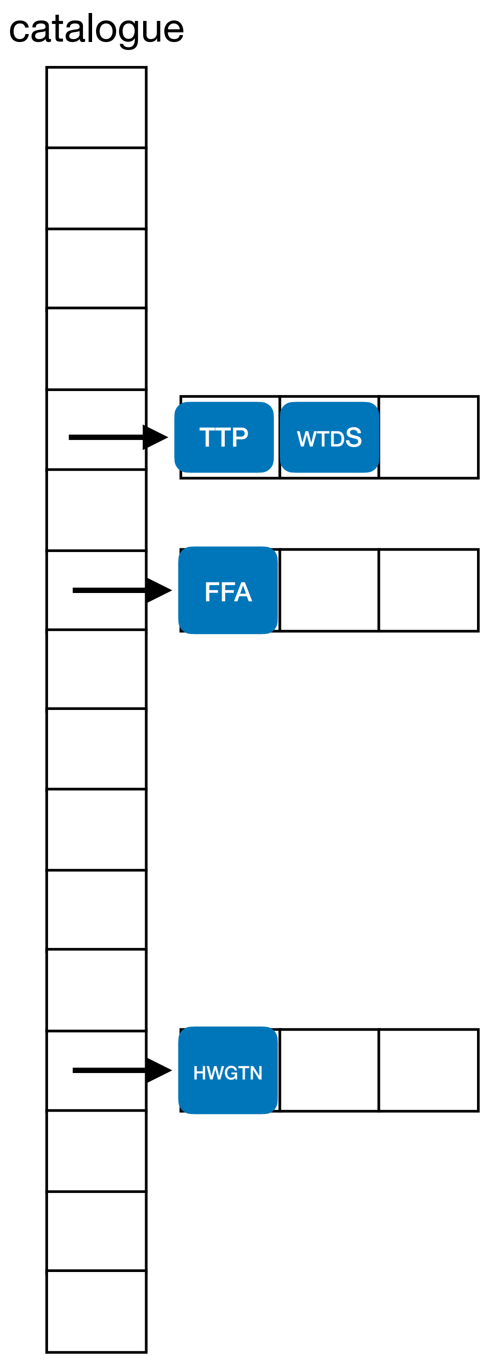
```java
import java.util.HashMap;
import java.util.List;
import java.util.Arrays;

public class Demo{
    public static void main(String[] args){
        HashMap<String,List<String>> catalogue = new HashMap<>();

        String k1 = "Malcolm Gladwell";
        String k2 = "Daniel Keys";
        String k3 = "Steven Johnson";

        String[] v1 = {"What the dog saw","The Tipping Point"};
        String[] v2 = {"Flowers for Algernon"};
        String[] v3 = {"How we got to now"};

        System.out.println(catalogue.put(k1,Arrays.asList(v1)));
        System.out.println(catalogue.put(k2,Arrays.asList(v2)));
        System.out.println(catalogue.put(k3,Arrays.asList(v3)));

        List<String> value;
        System.out.println("-------CATALOGUE LIST -----------");
        System.out.println("Catalogue: "+ catalogue.size() + " entries");
        for(String key: catalogue.keySet()){
            value = catalogue.get(key);
            prettyPrint(key, value);
        }
        System.out.println("---------------------------------");
    }

    public static void prettyPrint(String author, List<String> book){
        if(author.length() >15)
            System.out.println("Author: "+author + "\t Book(s): "+ book);
        else
            System.out.println("Author: "+author + "\t\t Book(s): "+ book);
    }
}
```



Example 2 - HashMap



```
null
null
null
-------CATALOGUE LIST -----------
Catalogue: 3 entries
Author: Steven Johnson          Book(s): [How we got to now]
Author: Daniel Keys             Book(s): [Flowers for Algernon]
Author: Malcolm Gladwell        Book(s): [What the dog saw, The Tipping Point]
---------------------------------
```



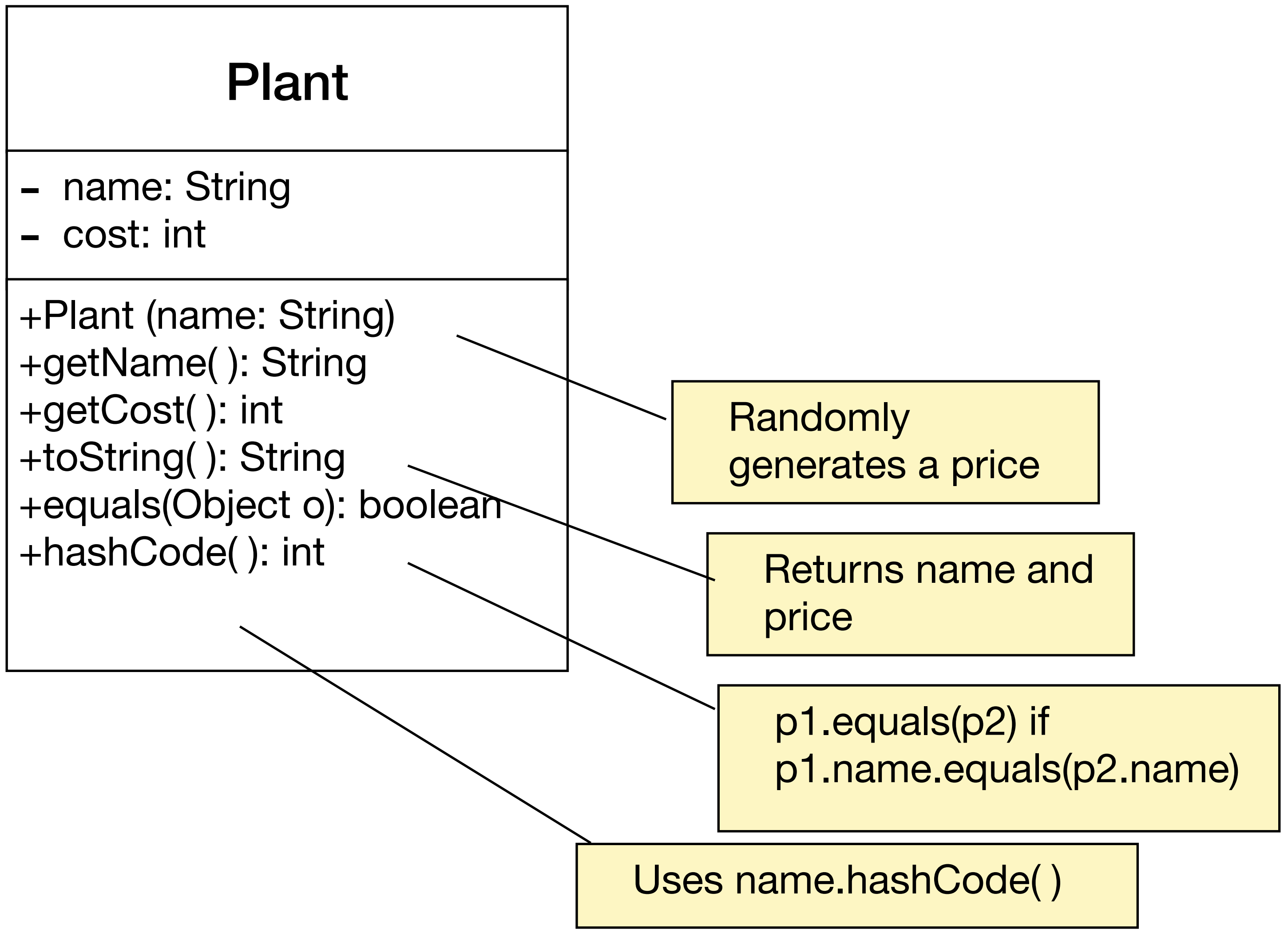
Another neat example of this nested approach can be found at:
https://viterbi-web.usc.edu/~adamchik/15-121/lectures/Hashing/code/Anagrams.java

That one demonstrates how a dictionary was used to create anagrams where it build a Map( ) whose key is a sorted word (meaning that its characters are sorted in alphabetical order) and whose values are the word's anagrams.

Full description here: https://viterbi-web.usc.edu/~adamchik/15-121/lectures/Hashing/hashing.html

Example 3 - HashMap

Suppose we use custom classes as the key and value.
Suppose we have a Plant class as follows:



Plant

- name: String
- cost: int

```
+Plant (name: String)
+getName( ): String
+getCost( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int
```

Randomly generates a price

Returns name and price

p1.equals(p2) if p1.name.equals(p2.name)

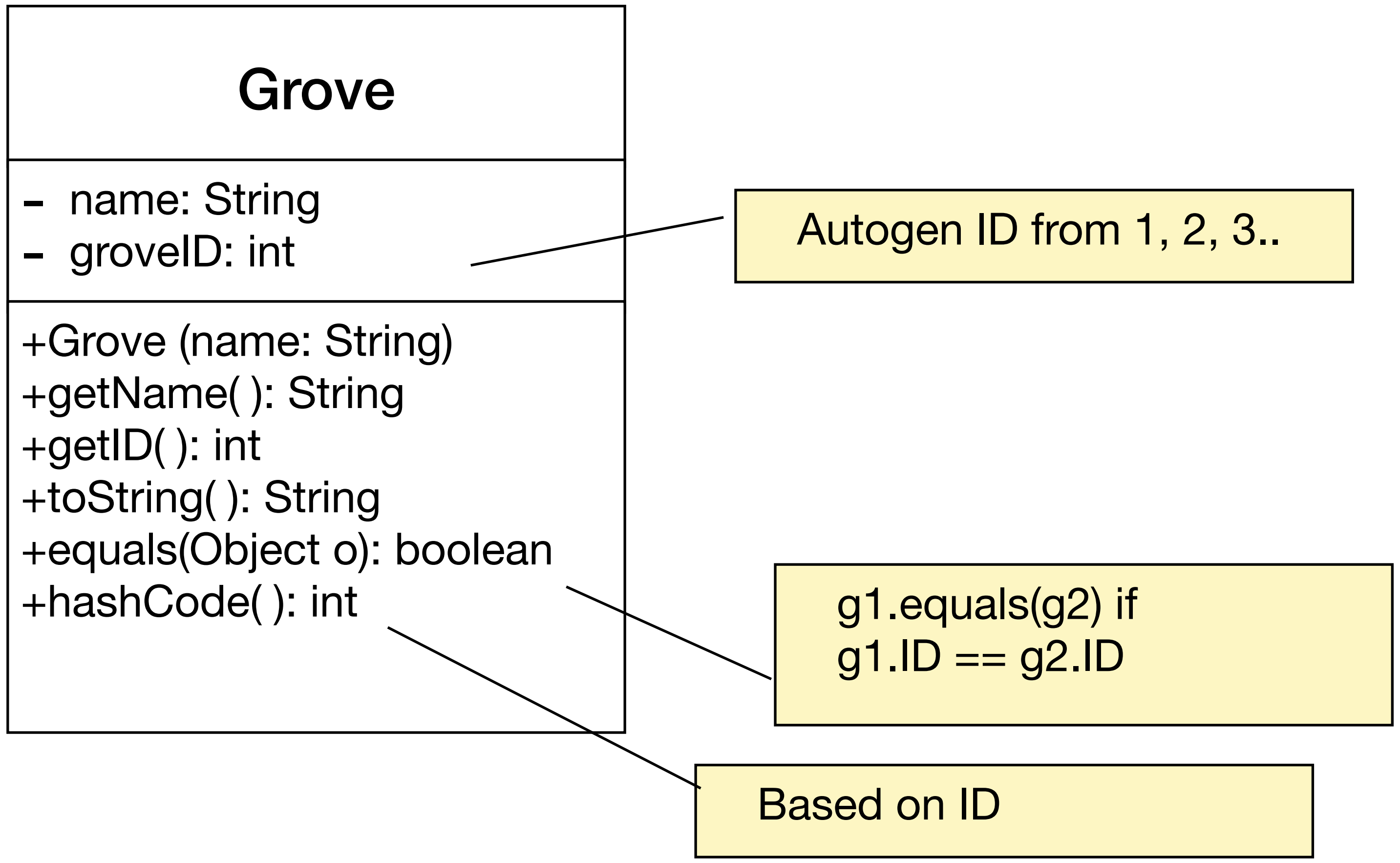
Uses name.hashCode( )

Example 3 - HashMap

Grove Class

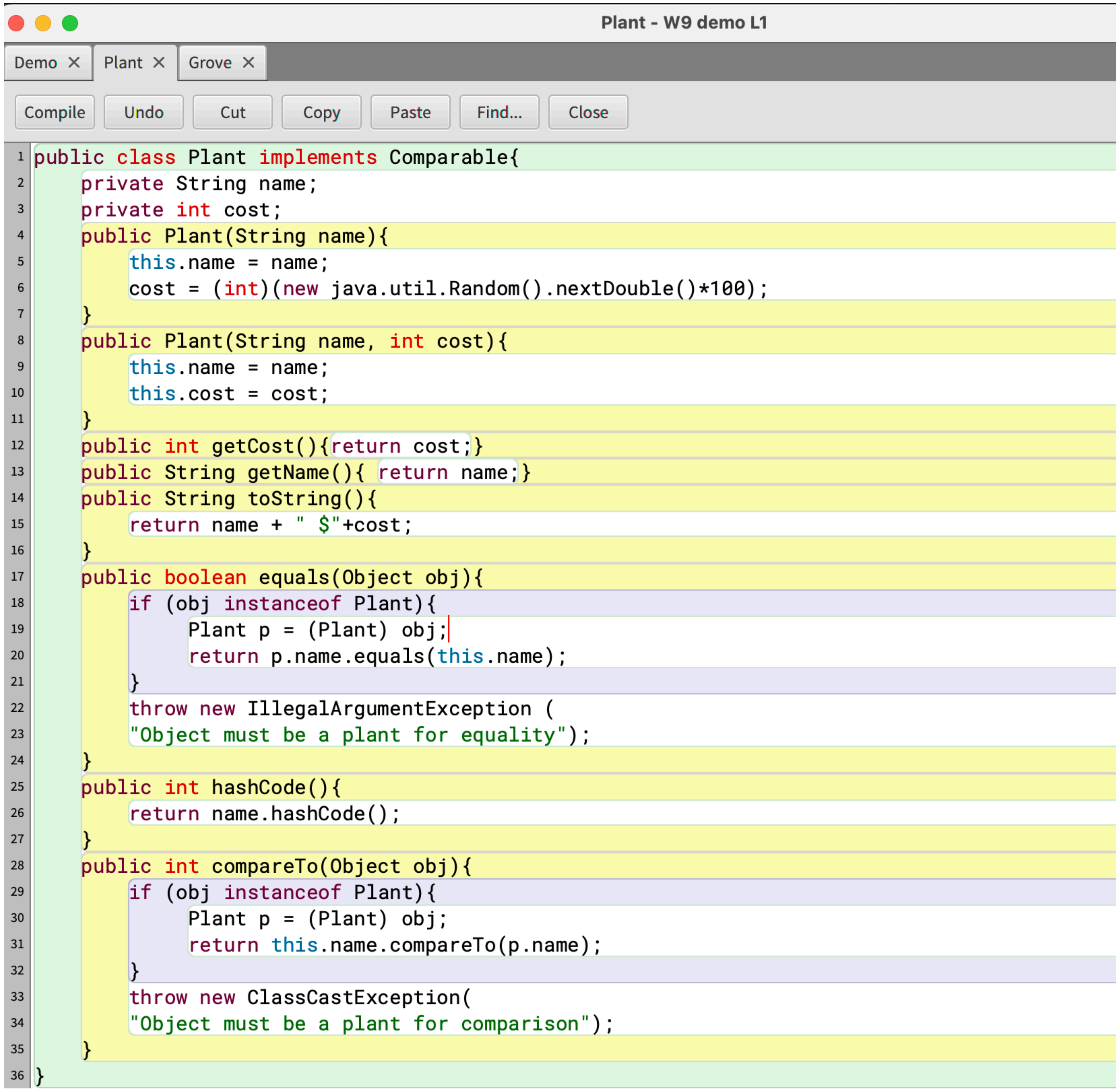
A Grove represents a group of specific types of plants

Pineapple Orange	Blood Orange	Navel Orange	Valencia Orange	Tangerine	Clementine
(Grove)	(Grove)	(Grove)	(Grove)	(Grove)	(Grove)



Example 3 - HashMap

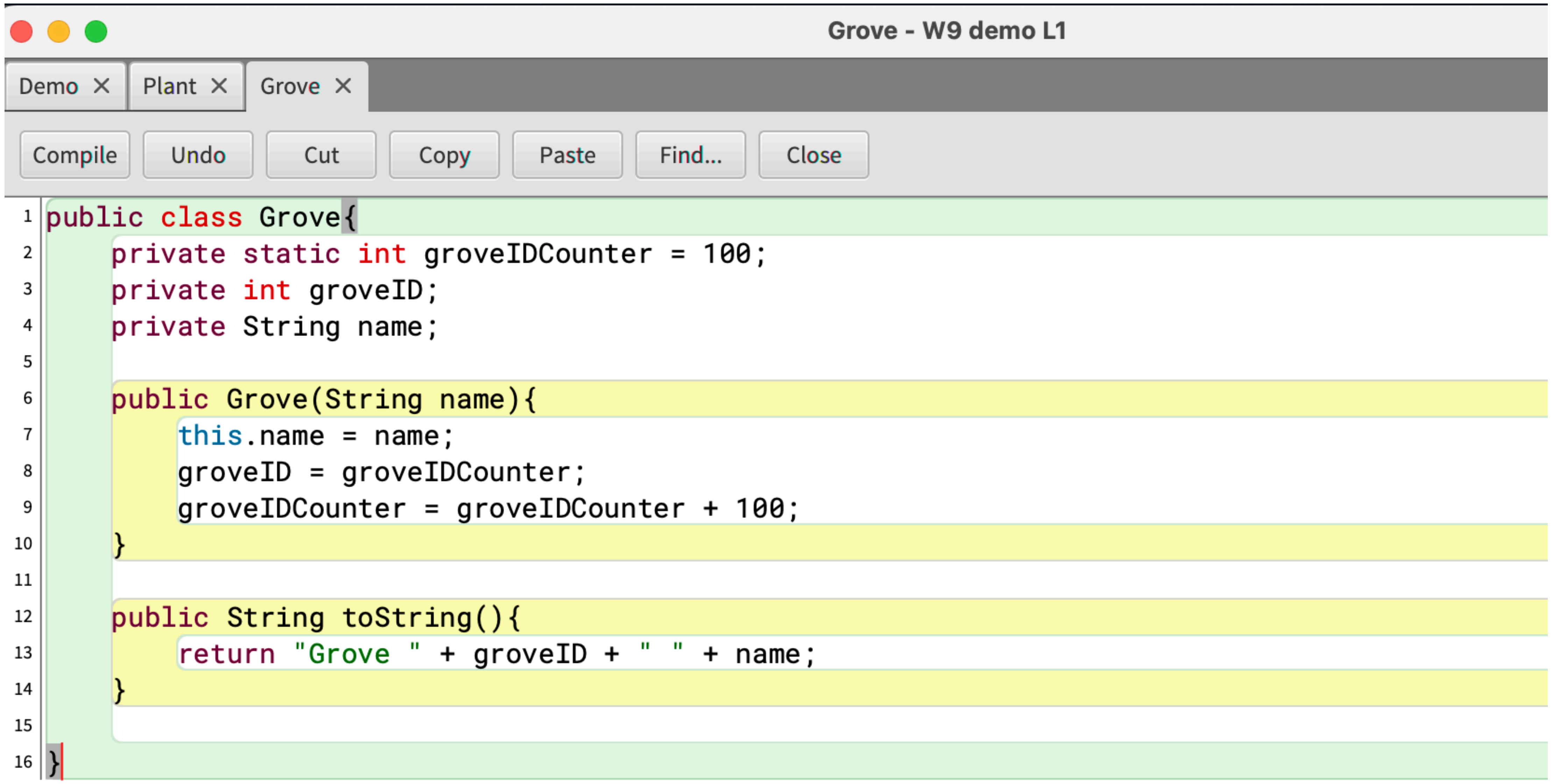
{key, value}
{Grove, Plant}



```java
public class Plant implements Comparable{
    private String name;
    private int cost;
    public Plant(String name){
        this.name = name;
        cost = (int)(new java.util.Random().nextDouble()*100);
    }
    public Plant(String name, int cost){
        this.name = name;
        this.cost = cost;
    }
    public int getCost(){return cost;}
    public String getName(){ return name;}
    public String toString(){
        return name + " $"+cost;
    }
    public boolean equals(Object obj){
        if (obj instanceof Plant){
            Plant p = (Plant) obj;
            return p.name.equals(this.name);
        }
        throw new IllegalArgumentException (
        "Object must be a plant for equality");
    }
    public int hashCode(){
        return name.hashCode();
    }
    public int compareTo(Object obj){
        if (obj instanceof Plant){
            Plant p = (Plant) obj;
            return this.name.compareTo(p.name);
        }
        throw new ClassCastException(
        "Object must be a plant for comparison");
    }
}
```

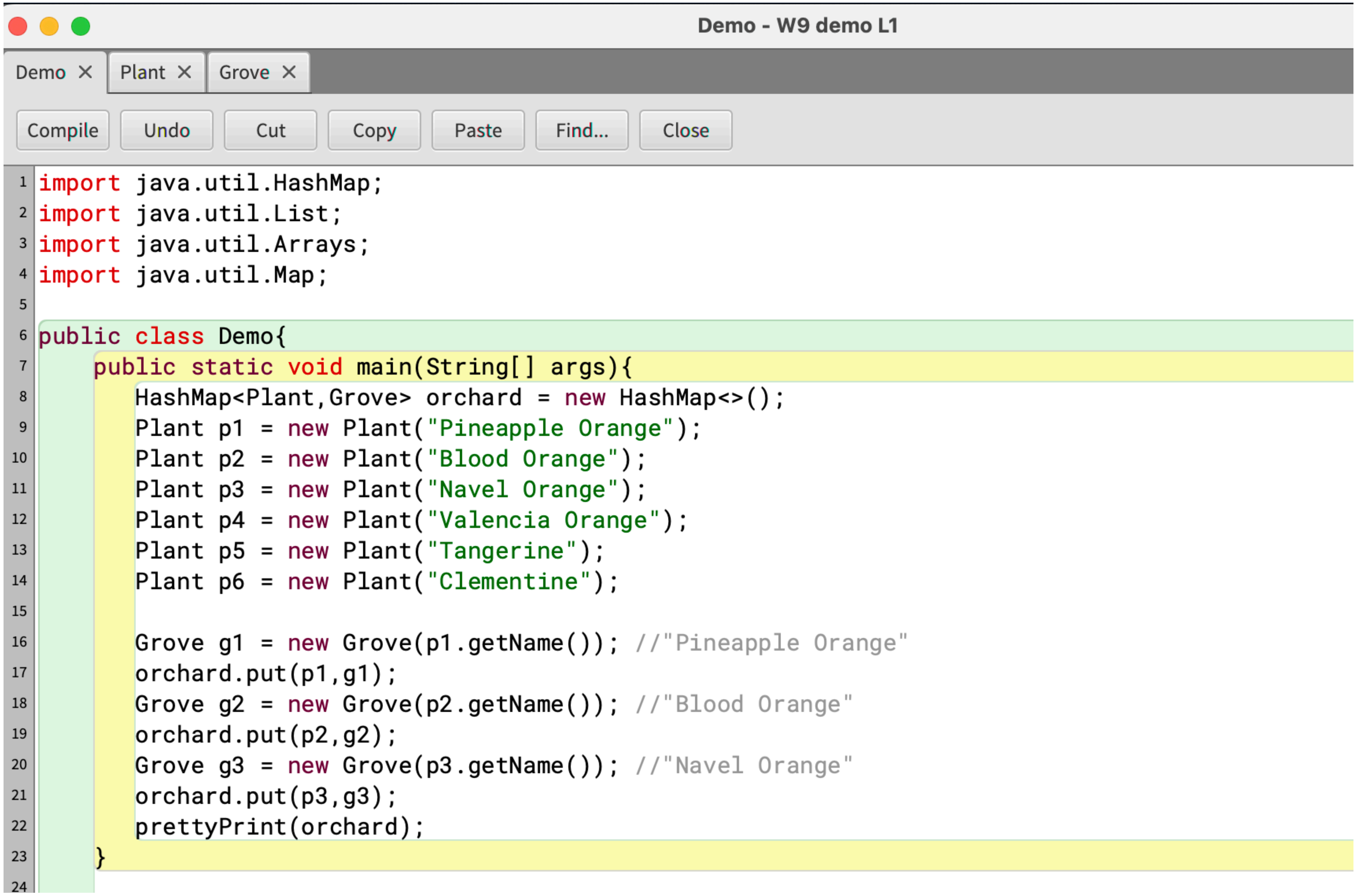
Example 3 - HashMap

{key, value}
{Grove, Plant}



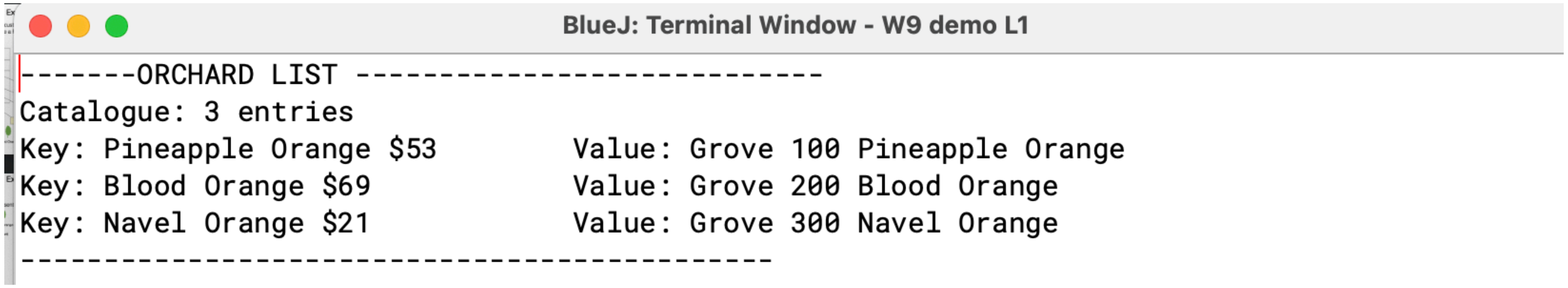
```java
public class Grove{
    private static int groveIDCounter = 100;
    private int groveID;
    private String name;

    public Grove(String name){
        this.name = name;
        groveID = groveIDCounter;
        groveIDCounter = groveIDCounter + 100;
    }

    public String toString(){
        return "Grove " + groveID + " " + name;
    }

}
```

Example 3 - HashMap



```java
import java.util.HashMap;
import java.util.List;
import java.util.Arrays;
import java.util.Map;

public class Demo{
    public static void main(String[] args){
        HashMap<Plant,Grove> orchard = new HashMap<>();
        Plant p1 = new Plant("Pineapple Orange");
        Plant p2 = new Plant("Blood Orange");
        Plant p3 = new Plant("Navel Orange");
        Plant p4 = new Plant("Valencia Orange");
        Plant p5 = new Plant("Tangerine");
        Plant p6 = new Plant("Clementine");

        Grove g1 = new Grove(p1.getName()); //"Pineapple Orange"
        orchard.put(p1,g1);
        Grove g2 = new Grove(p2.getName()); //"Blood Orange"
        orchard.put(p2,g2);
        Grove g3 = new Grove(p3.getName()); //"Navel Orange"
        orchard.put(p3,g3);
        prettyPrint(orchard);
    }
```

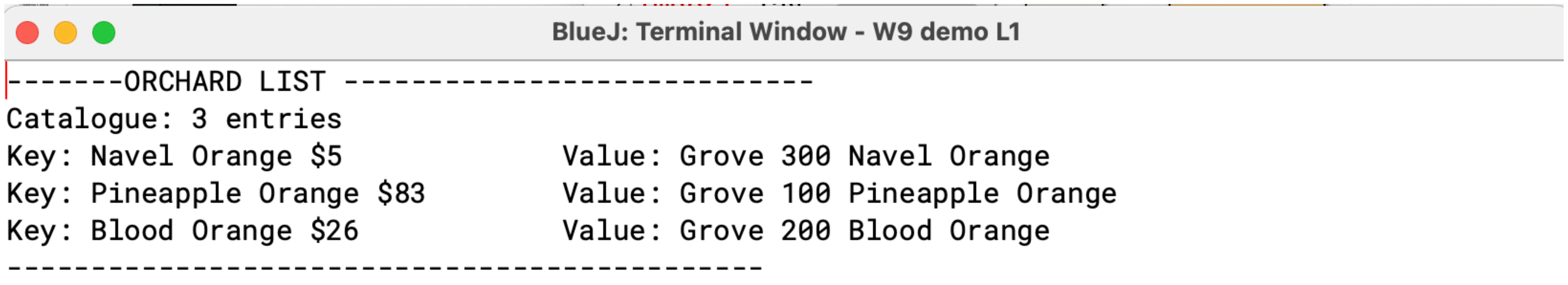
Example 3 - HashMap



BlueJ: Terminal Window - W9 demo L1

```
-------ORCHARD LIST ---------------------------
Catalogue: 3 entries
Key: Pineapple Orange $53        Value: Grove 100 Pineapple Orange
Key: Blood Orange $69            Value: Grove 200 Blood Orange
Key: Navel Orange $21            Value: Grove 300 Navel Orange
---------------------------------------------
```

Suppose we use remove the custom equals( ) and hashCode( ) from the Plant class



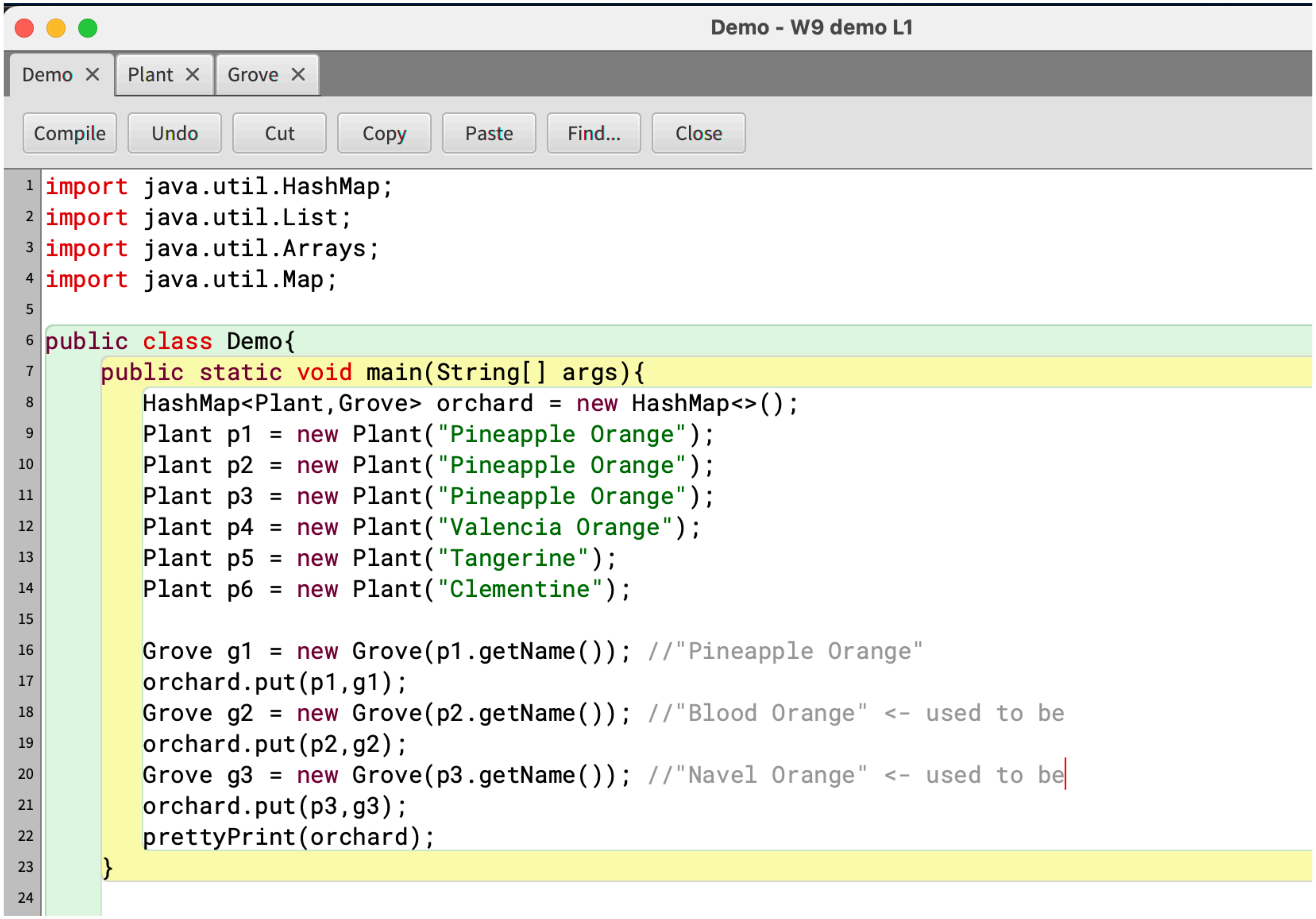
BlueJ: Terminal Window - W9 demo L1

```
-------ORCHARD LIST ---------------------------
Catalogue: 3 entries
Key: Navel Orange $5             Value: Grove 300 Navel Orange
Key: Pineapple Orange $83        Value: Grove 100 Pineapple Orange
Key: Blood Orange $26            Value: Grove 200 Blood Orange
---------------------------------------------
```

The code still works properly..

Example 3 - HashMap

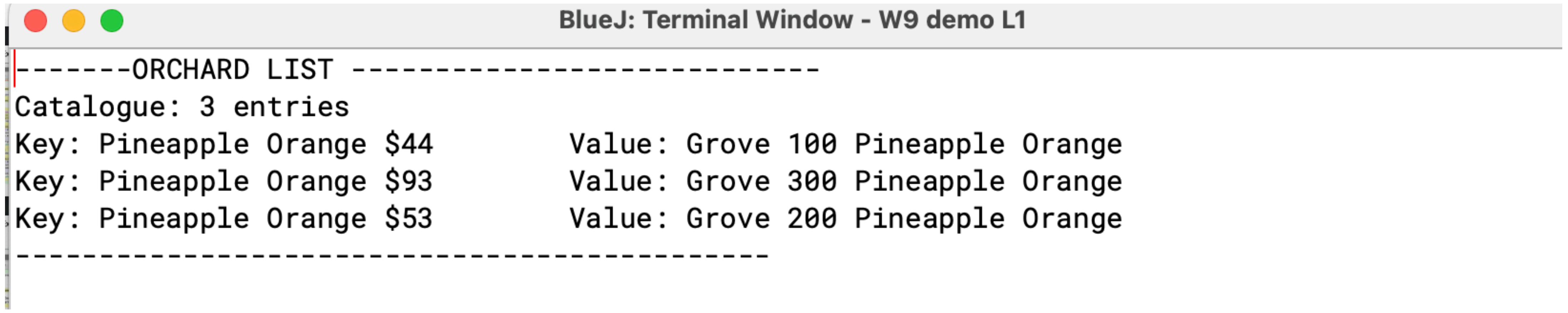
Suppose we modify the Plants as shown



```java
import java.util.HashMap;
import java.util.List;
import java.util.Arrays;
import java.util.Map;

public class Demo{
    public static void main(String[] args){
        HashMap<Plant,Grove> orchard = new HashMap<>();
        Plant p1 = new Plant("Pineapple Orange");
        Plant p2 = new Plant("Pineapple Orange");
        Plant p3 = new Plant("Pineapple Orange");
        Plant p4 = new Plant("Valencia Orange");
        Plant p5 = new Plant("Tangerine");
        Plant p6 = new Plant("Clementine");

        Grove g1 = new Grove(p1.getName()); //"Pineapple Orange"
        orchard.put(p1,g1);
        Grove g2 = new Grove(p2.getName()); //"Blood Orange" <- used to be
        orchard.put(p2,g2);
        Grove g3 = new Grove(p3.getName()); //"Navel Orange" <- used to be
        orchard.put(p3,g3);
        prettyPrint(orchard);
    }

```

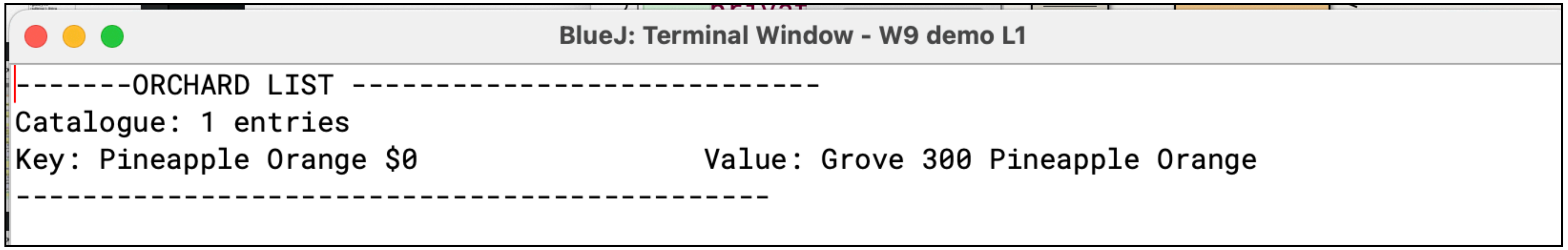
Example 3 - HashMap

The Plant objects are still all unique because of the memory locations -> HashMap insertion in unaffected although this is a conceptual error



```
-------ORCHARD LIST ---------------------------
Catalogue: 3 entries
Key: Pineapple Orange $44        Value: Grove 100 Pineapple Orange
Key: Pineapple Orange $93        Value: Grove 300 Pineapple Orange
Key: Pineapple Orange $53        Value: Grove 200 Pineapple Orange
---------------------------------------------
```

Restoring the equals( ) and hashCode( ) in the Plant results in the correct behaviour. Note that the last Grove object replaces all others previously stored.



```
-------ORCHARD LIST ---------------------------
Catalogue: 1 entries
Key: Pineapple Orange $0                Value: Grove 300 Pineapple Orange
---------------------------------------------
```

TreeMap

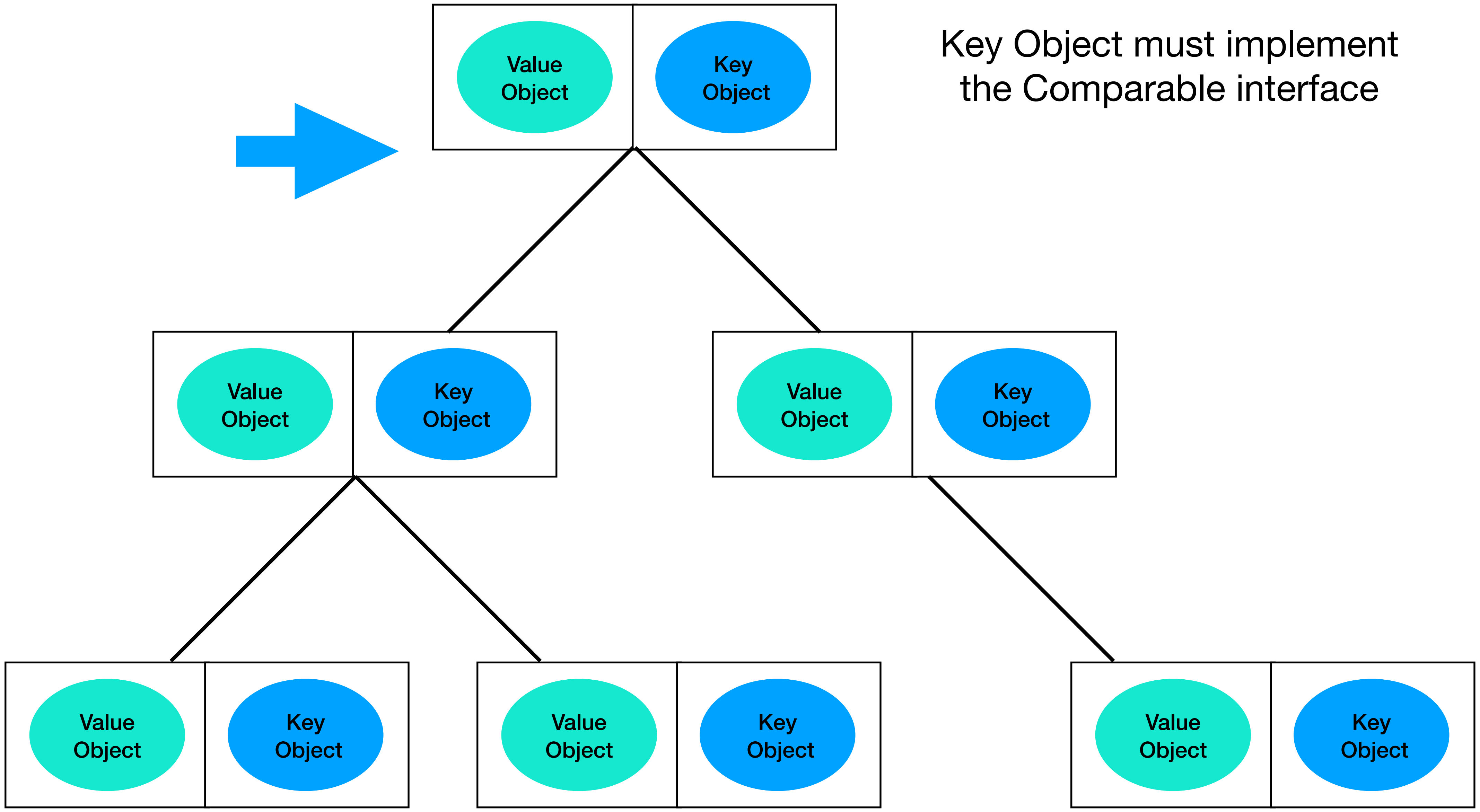
A Red-Black tree based NavigableMap implementation. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

Note that the **ordering maintained by a tree map**, like any sorted map, and whether or not an explicit comparator is provided, must be **consistent with equals** if this sorted map is to correctly implement the Map interface. (See Comparable or Comparator for a precise definition of consistent with equals.)

This is so because the Map interface is defined in terms of the equals operation, but a sorted map performs all key comparisons using its **compareTo (or compare) method**, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/TreeMap.html

TreeMap



Key Object must implement the Comparable interface

TreeMap

Creating and instantiating:

Key type **Value type**

```
TreeMap < A, B > tm;                // declaration
tm = new TreeMap < A, B >(); // initialisation
```

The key must implement the Comparable interface

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/TreeMap.html

TreeMap

Creating and instantiating:

Key type **Value type**

```java
TreeMap <String,Plant> plants; // declaration
plants = new TreeMap<>(); // initialisation
```

```java
Map<String,Plant> herbs; // declaration as Polymorphic obj
herbs = new TreeMap<>(); // new Tree Map – dynamic type
```

The Sorted Map Interface

The SortedMap interface represents a Map object that keeps its set of key objects in sorted order. Its keySet( ) and values( ) methods inherited from Map return collection that can be traversed in sorted order of the key.

It also declares methods of its own such as firstKey( ) and lastKey() that return the lowest and highest key values in the SortedMap.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/SortedMap.html

Comparable Interface

This interface imposes a total ordering on the objects of each class that implements it.

This ordering is referred to as the class's natural ordering, and the class's **compareTo** method is referred to as its natural comparison method.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Comparable.html

int compareTo(Object obj)

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

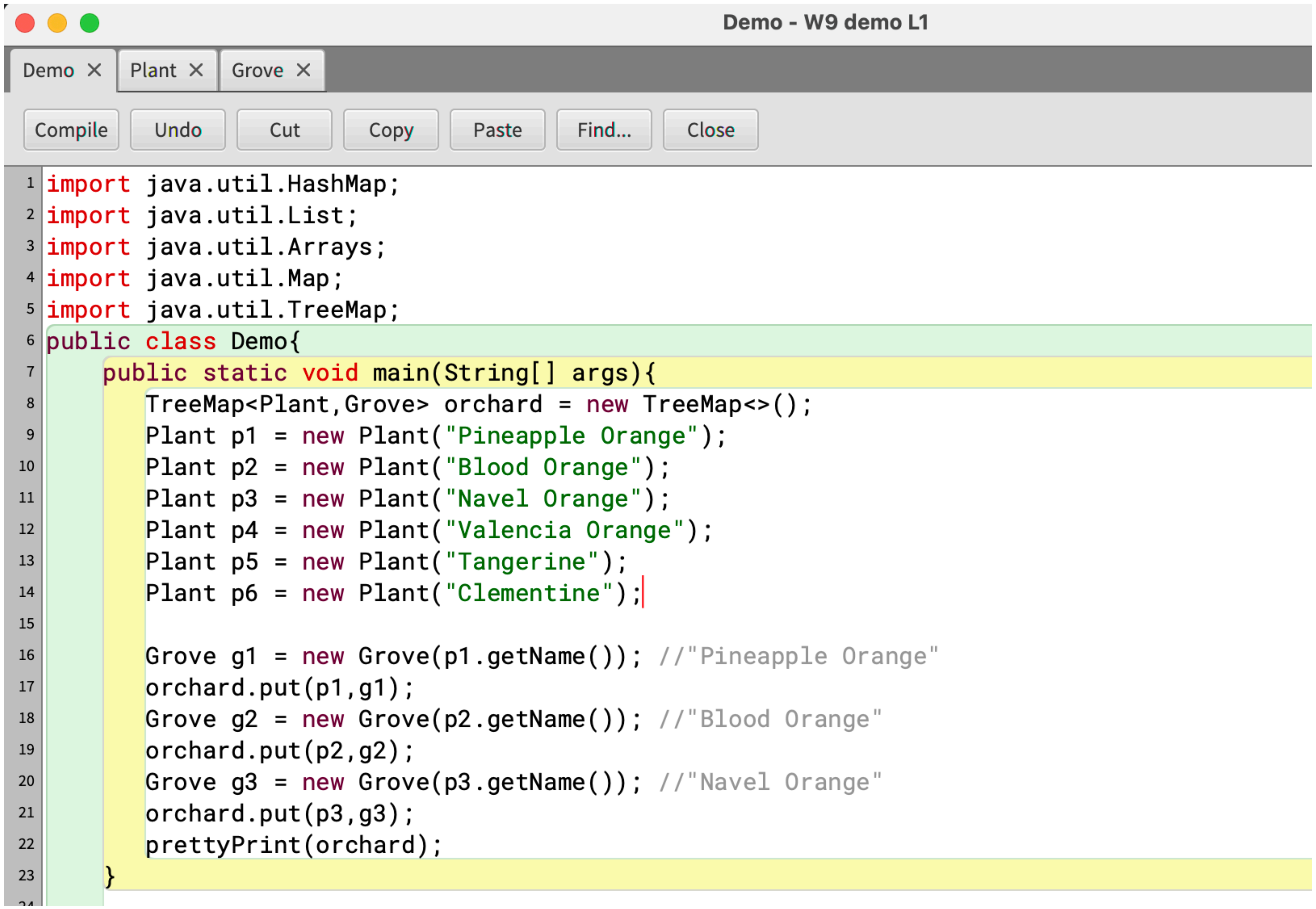
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Comparable.html#compareTo(T)

Comparable Interface

The natural ordering for a class C is said to be consistent with equals if and only if e1.compareTo(e2) == 0 has the same boolean value as e1.equals(e2) for every e1 and e2 of class C.

Note that null is not an instance of any class, and e.compareTo(null) should throw a NullPointerException even though e.equals(null) returns false.

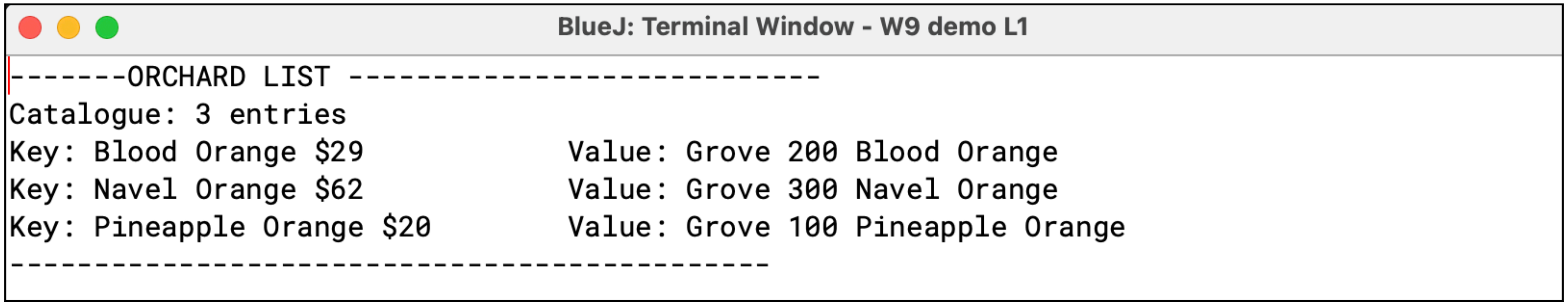
Example 4 - TreeMap



```
import java.util.HashMap;
import java.util.List;
import java.util.Arrays;
import java.util.Map;
import java.util.TreeMap;
public class Demo{
    public static void main(String[] args){
        TreeMap<Plant,Grove> orchard = new TreeMap<>();
        Plant p1 = new Plant("Pineapple Orange");
        Plant p2 = new Plant("Blood Orange");
        Plant p3 = new Plant("Navel Orange");
        Plant p4 = new Plant("Valencia Orange");
        Plant p5 = new Plant("Tangerine");
        Plant p6 = new Plant("Clementine");

        Grove g1 = new Grove(p1.getName()); //"Pineapple Orange"
        orchard.put(p1,g1);
        Grove g2 = new Grove(p2.getName()); //"Blood Orange"
        orchard.put(p2,g2);
        Grove g3 = new Grove(p3.getName()); //"Navel Orange"
        orchard.put(p3,g3);
        prettyPrint(orchard);
    }
```

Suppose we use a TreeMap to store the objects.

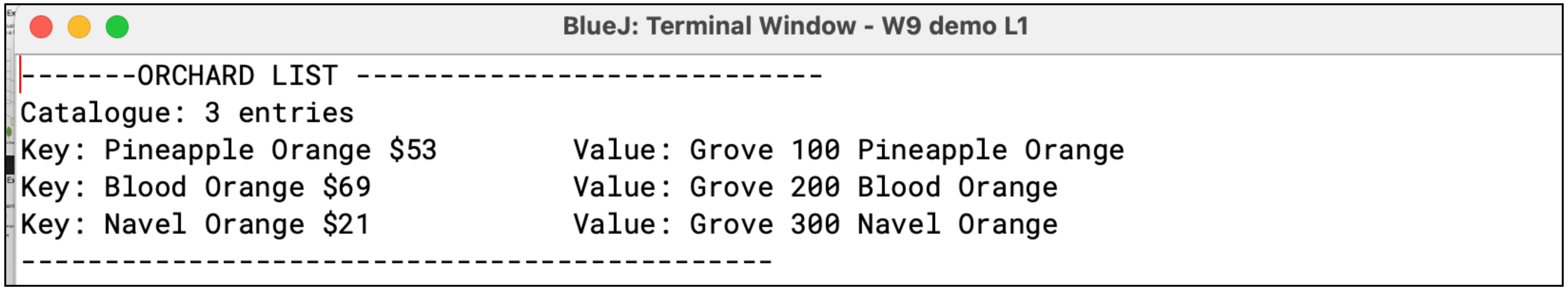
Example 4 - TreeMap



```
-------ORCHARD LIST ----------------------------
Catalogue: 3 entries
Key: Blood Orange $29           Value: Grove 200 Blood Orange
Key: Navel Orange $62           Value: Grove 300 Navel Orange
Key: Pineapple Orange $20       Value: Grove 100 Pineapple Orange
---------------------------------------------
```

The keys are sorted alphabetically. This happens because the Plant implements the Comparable interface

Example 3 - HashMap

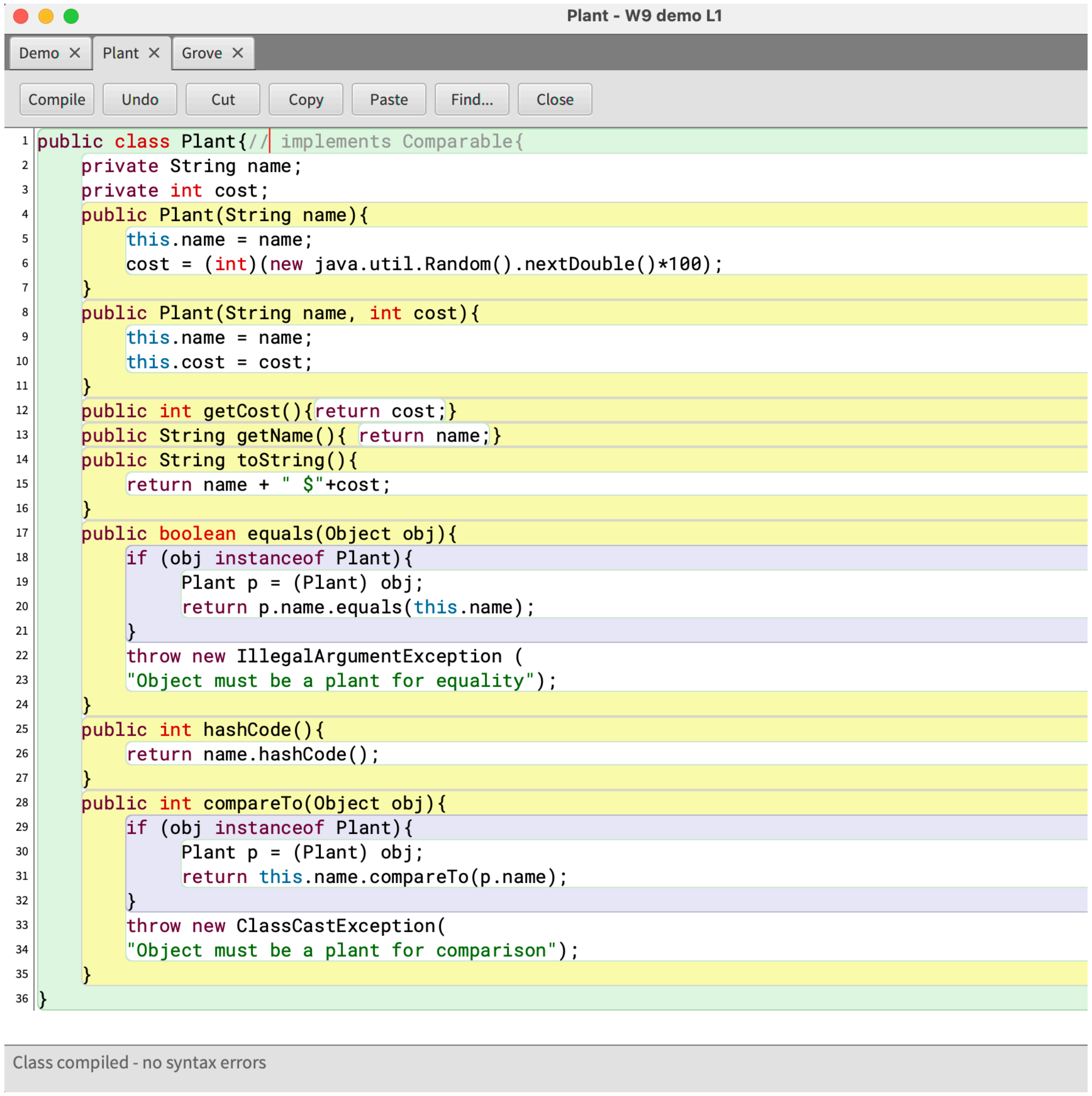


```
-------ORCHARD LIST ----------------------------
Catalogue: 3 entries
Key: Pineapple Orange $53       Value: Grove 100 Pineapple Orange
Key: Blood Orange $69           Value: Grove 200 Blood Orange
Key: Navel Orange $21           Value: Grove 300 Navel Orange
---------------------------------------------
```

Compare with the HashMap result: the keys are not ordered

Example 5 - TreeMap

Suppose we remove the subtyping of the Comparable interface in the Plant class

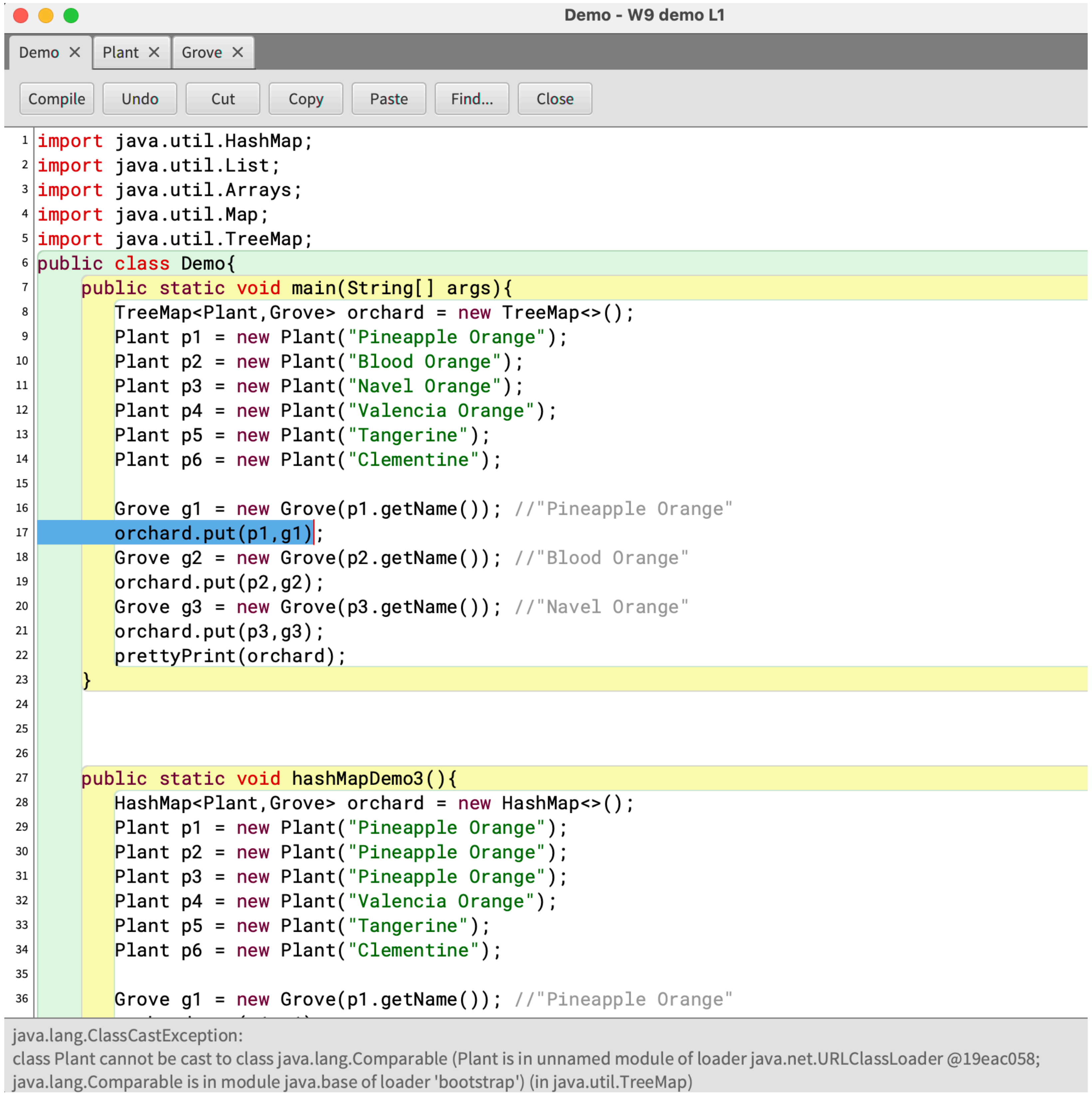


```java
public class Plant{// implements Comparable{
    private String name;
    private int cost;
    public Plant(String name){
        this.name = name;
        cost = (int)(new java.util.Random().nextDouble()*100);
    }
    public Plant(String name, int cost){
        this.name = name;
        this.cost = cost;
    }
    public int getCost(){return cost;}
    public String getName(){ return name;}
    public String toString(){
        return name + " $"+cost;
    }
    public boolean equals(Object obj){
        if (obj instanceof Plant){
             Plant p = (Plant) obj;
             return p.name.equals(this.name);
        }
        throw new IllegalArgumentException (
        "Object must be a plant for equality");
    }
    public int hashCode(){
        return name.hashCode();
    }
    public int compareTo(Object obj){
        if (obj instanceof Plant){
             Plant p = (Plant) obj;
             return this.name.compareTo(p.name);
        }
        throw new ClassCastException(
        "Object must be a plant for comparison");
    }
}
```

Class compiled - no syntax errors

Example 5 - TreeMap

The code will compile but an error is thrown at runtime



```java
import java.util.HashMap;
import java.util.List;
import java.util.Arrays;
import java.util.Map;
import java.util.TreeMap;
public class Demo{
    public static void main(String[] args){
        TreeMap<Plant,Grove> orchard = new TreeMap<>();
        Plant p1 = new Plant("Pineapple Orange");
        Plant p2 = new Plant("Blood Orange");
        Plant p3 = new Plant("Navel Orange");
        Plant p4 = new Plant("Valencia Orange");
        Plant p5 = new Plant("Tangerine");
        Plant p6 = new Plant("Clementine");

        Grove g1 = new Grove(p1.getName()); //"Pineapple Orange"
        orchard.put(p1,g1);
        Grove g2 = new Grove(p2.getName()); //"Blood Orange"
        orchard.put(p2,g2);
        Grove g3 = new Grove(p3.getName()); //"Navel Orange"
        orchard.put(p3,g3);
        prettyPrint(orchard);
    }



    public static void hashMapDemo3(){
        HashMap<Plant,Grove> orchard = new HashMap<>();
        Plant p1 = new Plant("Pineapple Orange");
        Plant p2 = new Plant("Pineapple Orange");
        Plant p3 = new Plant("Pineapple Orange");
        Plant p4 = new Plant("Valencia Orange");
        Plant p5 = new Plant("Tangerine");
        Plant p6 = new Plant("Clementine");

        Grove g1 = new Grove(p1.getName()); //"Pineapple Orange"
```

```
java.lang.ClassCastException:
class Plant cannot be cast to class java.lang.Comparable (Plant is in unnamed module of loader java.net.URLClassLoader @19eac058;
java.lang.Comparable is in module java.base of loader 'bootstrap') (in java.util.TreeMap)
```

Summary

Today you learned about:

- Concrete Maps: HashMap, TreeMap
- The differences in how Maps are manipulated compared to Collections
  - Stores {key,value} pairs
  - Must use Objects for keys and values
  - Keys are used for insertion into maps (hashCode)
  - Different method (put, values( ), keySet( ) etc)
  - Iteration uses keys (or values)
  - Keys are used to retrieve values via get(..)
- Nested collections
  - Using another collection as a value in a Map
- Sorted Maps: TreeMap
  - Key must implement Comparable