Container Classes

Concrete Collections: TreeMap

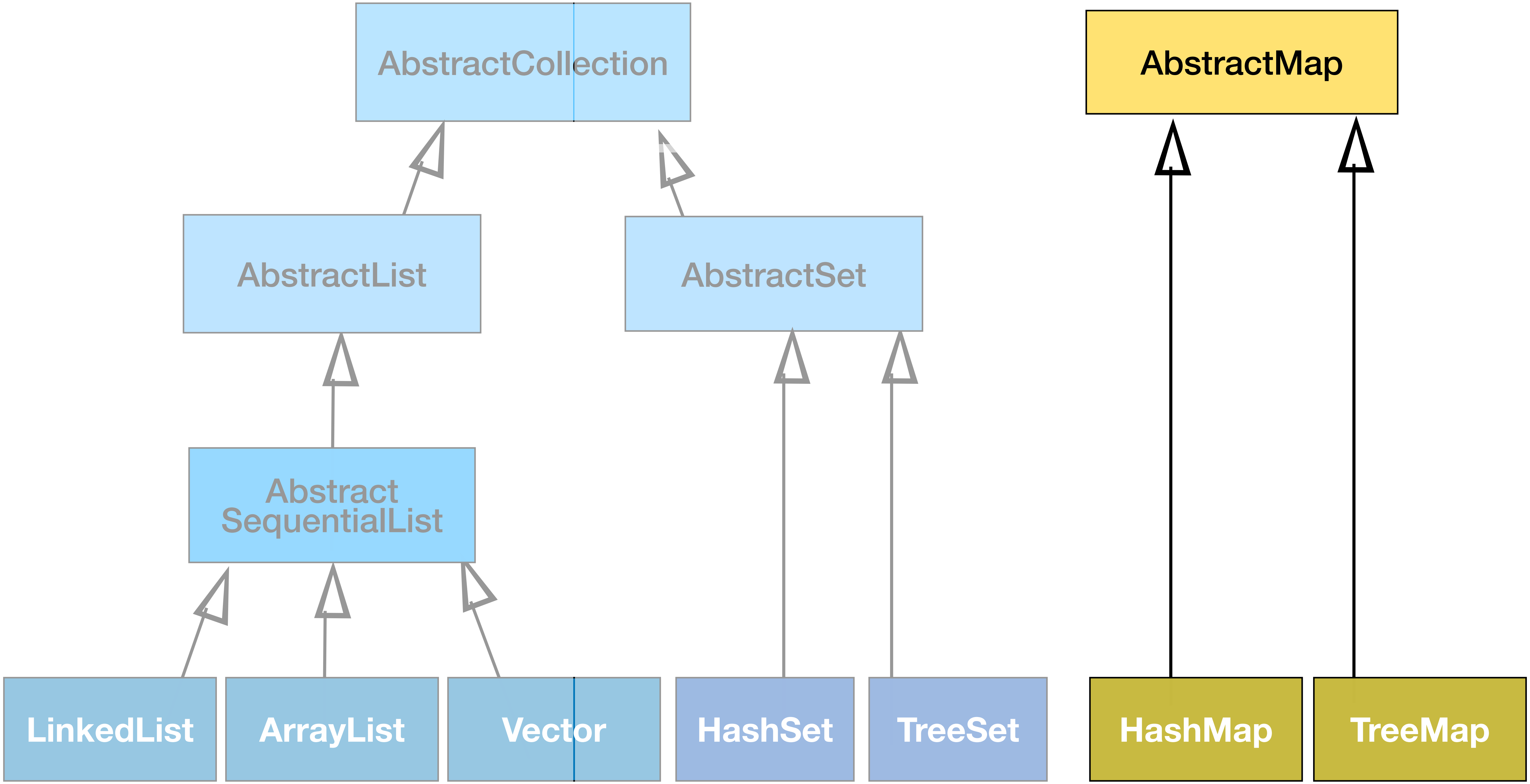
COMP2603
Object Oriented Programming 1

Week 12

Outline

- Concrete Collections
  - TreeMap and HashMap
    - Creation
    - Adding elements
    - Finding elements
    - Removing elements
    - Traversal

Classes in the Java Collections Framework



HashMap

(K, V)

Hash table based implementation of the Map interface.

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/12/docs/api/java.base/java/util/HashMap.html

Comparable → compareTo(1)

-1 0 1

TreeMap

(K, V)

A TreeMap is built as a Red-Black tree based **NavigableMap** implementation.

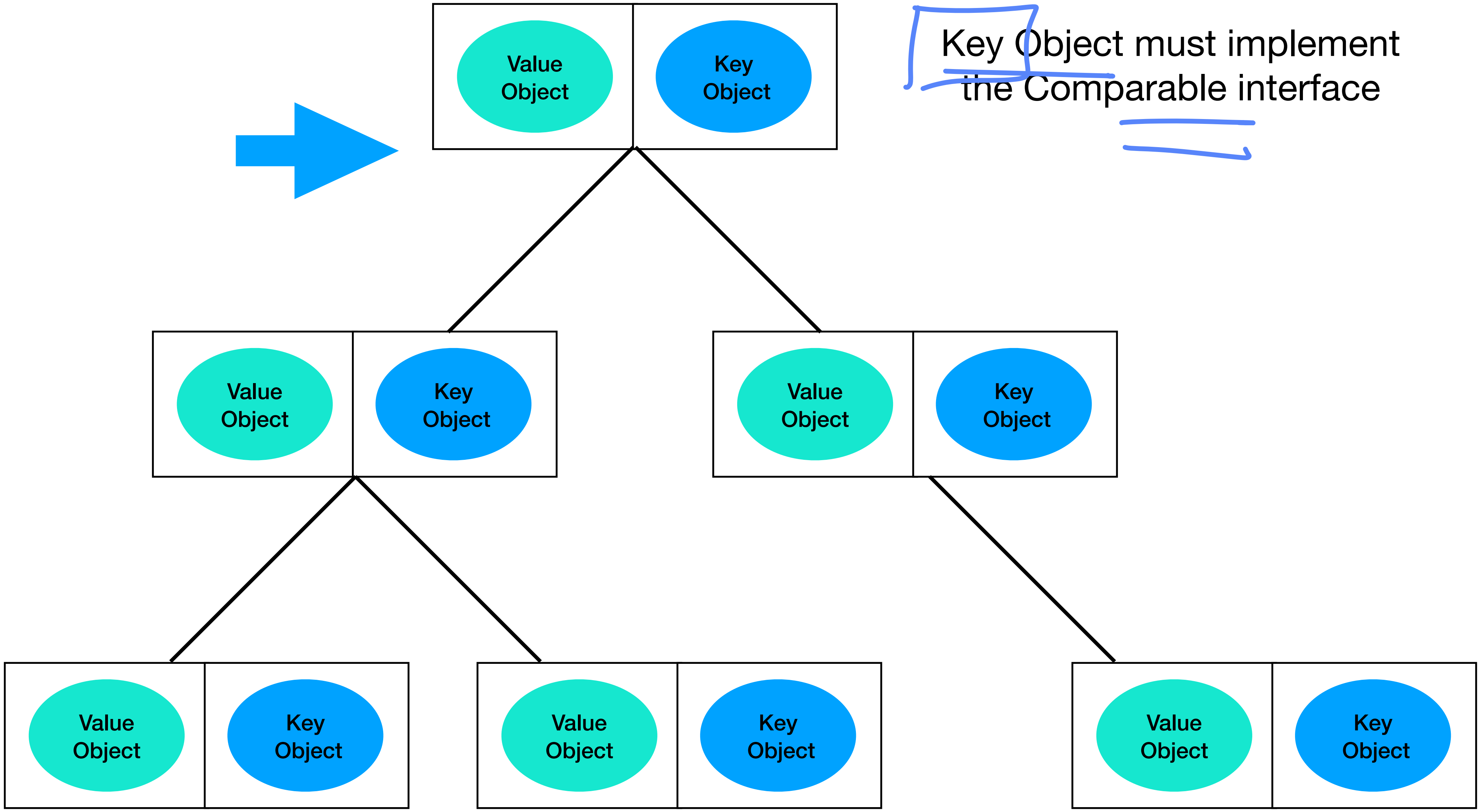
The map is sorted according to the natural ordering of its **keys**, or by a **Comparator** provided at map creation time, depending on which constructor is used.

interface

compare(Object o1, Object o2)

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

TreeMap



Key Object must implement the Comparable interface

TreeMap

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*.)

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

TreeMap

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.

The behaviour of a sorted map *is* well-defined even if its ordering is inconsistent with **`equals`**; it just fails to obey the general contract of the **`Map`** interface.

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

TreeMap

Creating and instantiating:

Key type **Value type**

```
TreeMap < A, B > tm;                // declaration    1.
tm = new TreeMap < A, B >(); // initialisation   2.
```

The key must implement the Comparable interface

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

TreeMap

Creating and instantiating:

Key type **Value type**

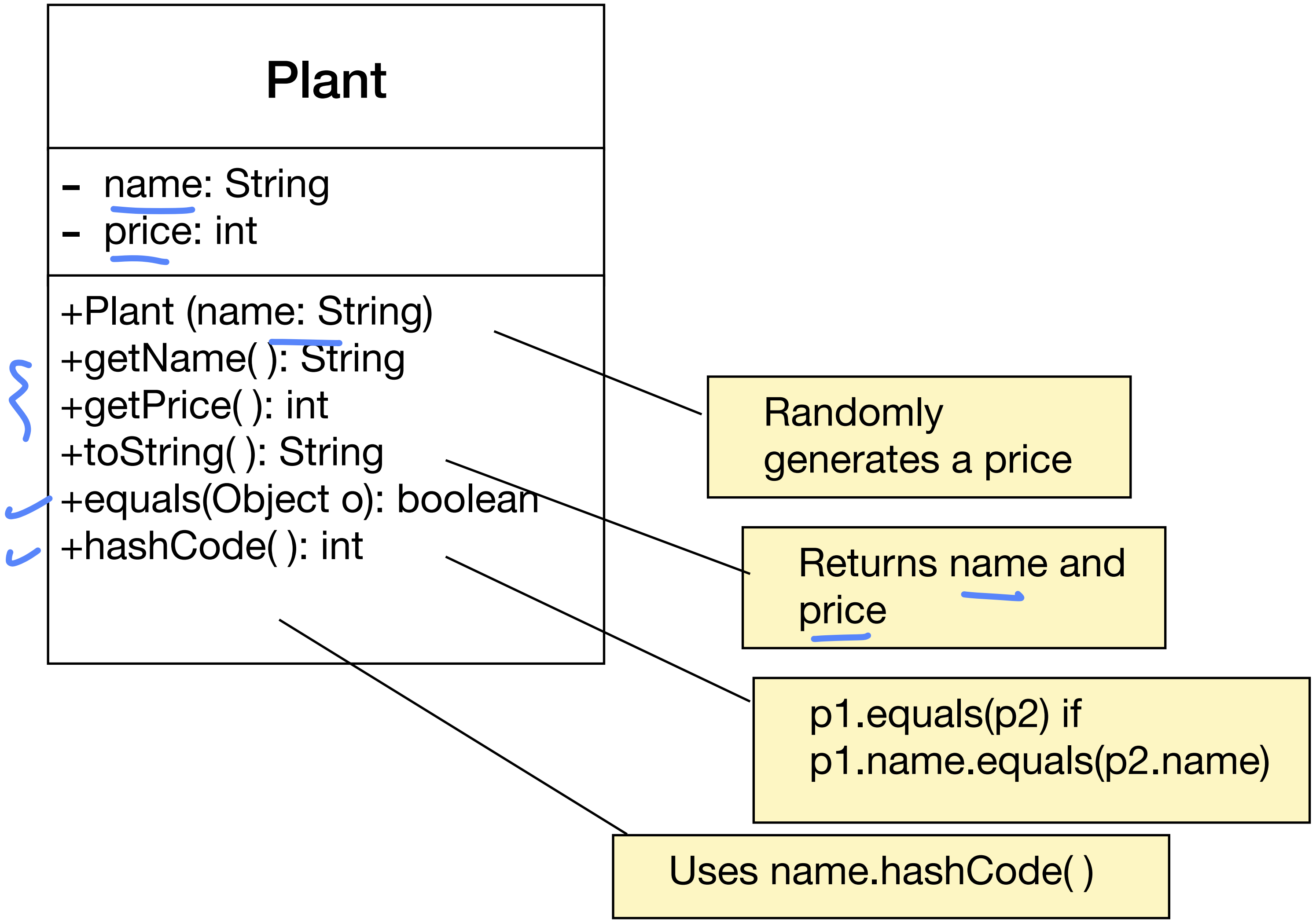
```
1. TreeMap <String,Plant> plants; // declaration
2. plants = new TreeMap<>(); // initialisation
```

```
3. Map<String,Plant> herbs; // declaration as Polymorphic obj
4. herbs = new TreeMap<>(); // new Tree Map - dynamic type
```

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

Plant Class

Suppose we have a Plant class as follows:



Plant

- name: String
- price: int

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

Randomly generates a price

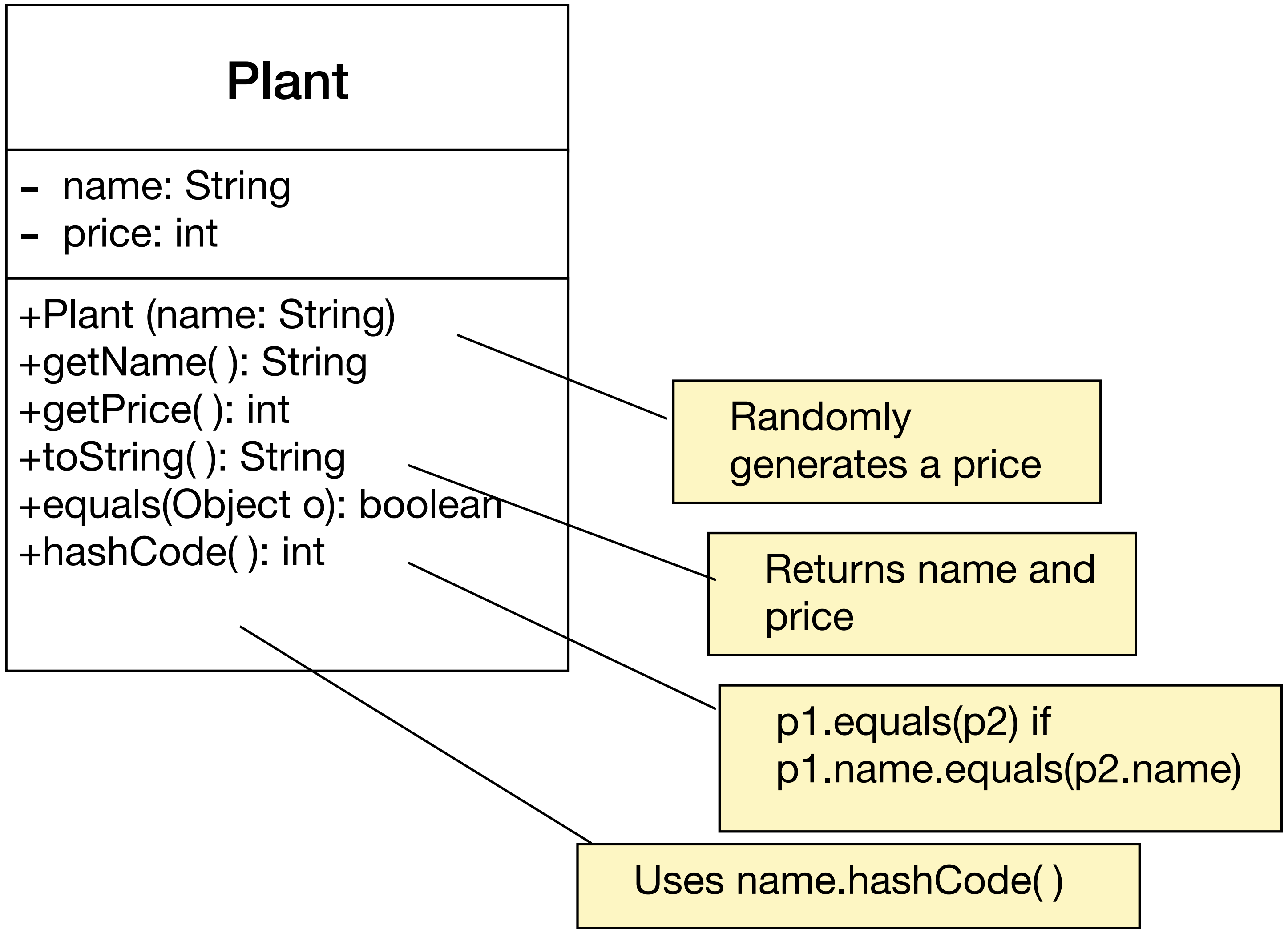
Returns name and price

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

Uses name.hashCode( )

Plant Class

Suppose we have a Plant class as follows:



Plant

- name: String
- price: int

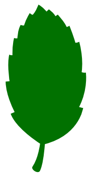
+Plant (name: String)
+getName( ): String
+getPrice( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int

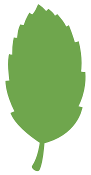
Randomly generates a price

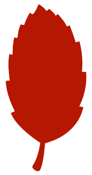
Returns name and price

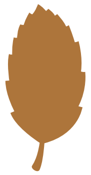
p1.equals(p2) if p1.name.equals(p2.name)

Uses name.hashCode( )


Pineapple Orange


Blood Orange


Navel Orange


Valencia Orange

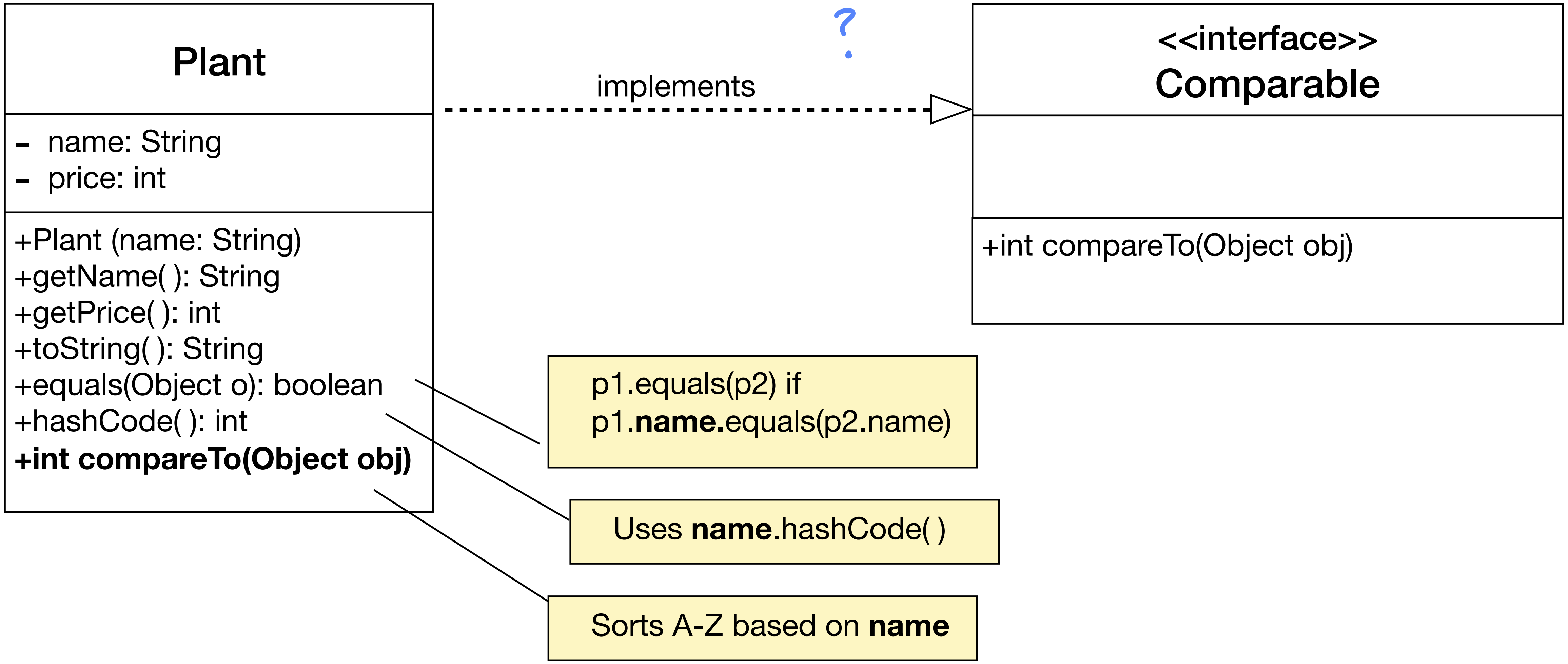
Tangerine

Clementine

Plant Class

Suppose we have a Plant class as follows:

TreeMap
↓ if Plant = key



Plant

- name: String
- price: int

+Plant (name: String)
+getName( ): String
+getPrice( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int
+int compareTo(Object obj)

implements

?

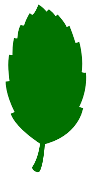
<<interface>>
Comparable

+int compareTo(Object obj)

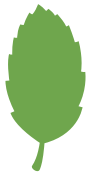
p1.equals(p2) if
p1.**name.**equals(p2.name)

Uses **name**.hashCode( )

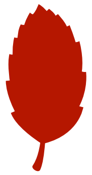
Sorts A-Z based on **name**



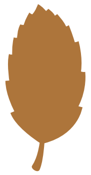
Pineapple Orange



Blood Orange



Navel Orange



Valencia Orange

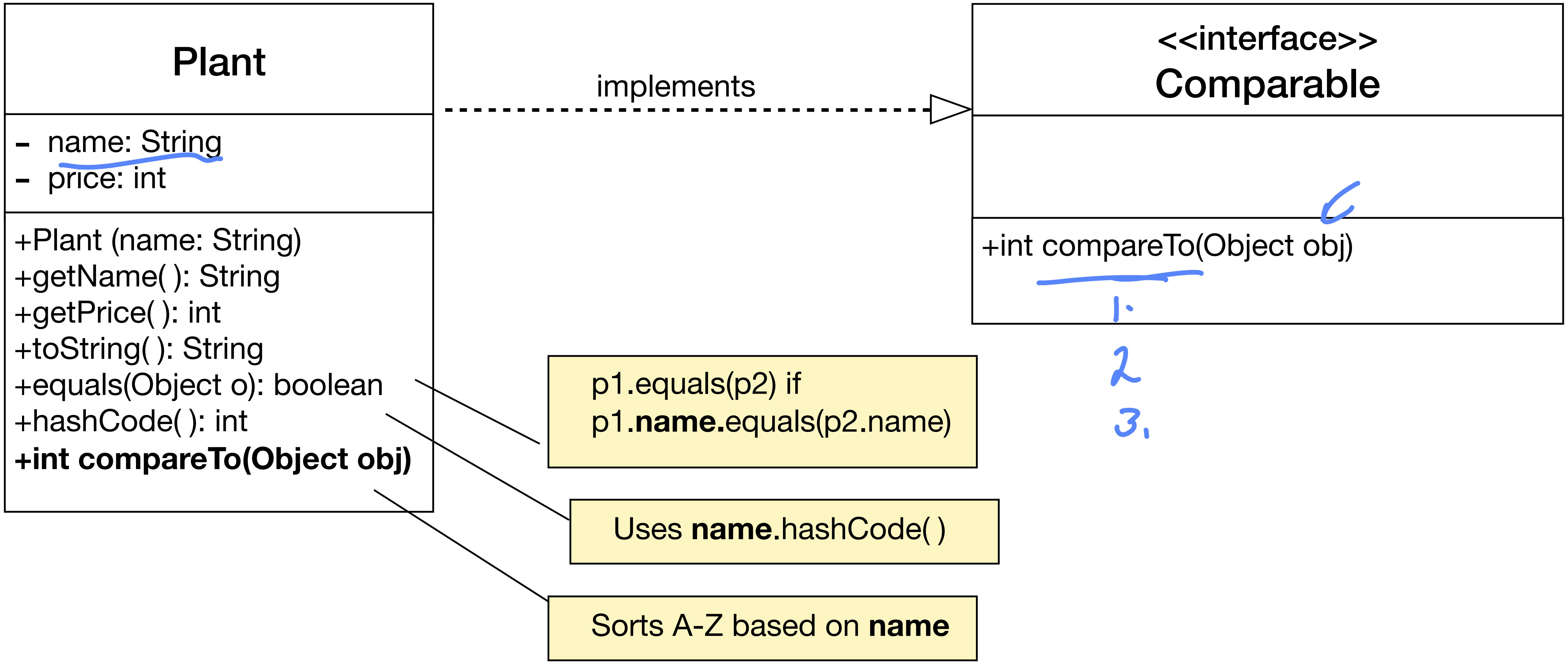
Tangerine

Clementine

Plant Class

TreeSet < >

Suppose we have a Plant class as follows:



Plant

- name: String
- price: int

+Plant (name: String)
+getName( ): String
+getPrice( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int
+int compareTo(Object obj)

implements

<<interface>>
Comparable

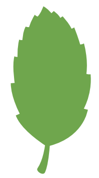
+int compareTo(Object obj)

1.
2
3.

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

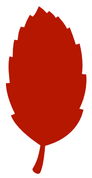
Uses **name**.hashCode( )

Sorts A-Z based on **name**

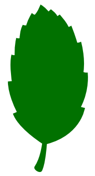


Blood Orange

Clementine

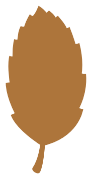


Navel Orange



Pineapple Orange

Tangerine

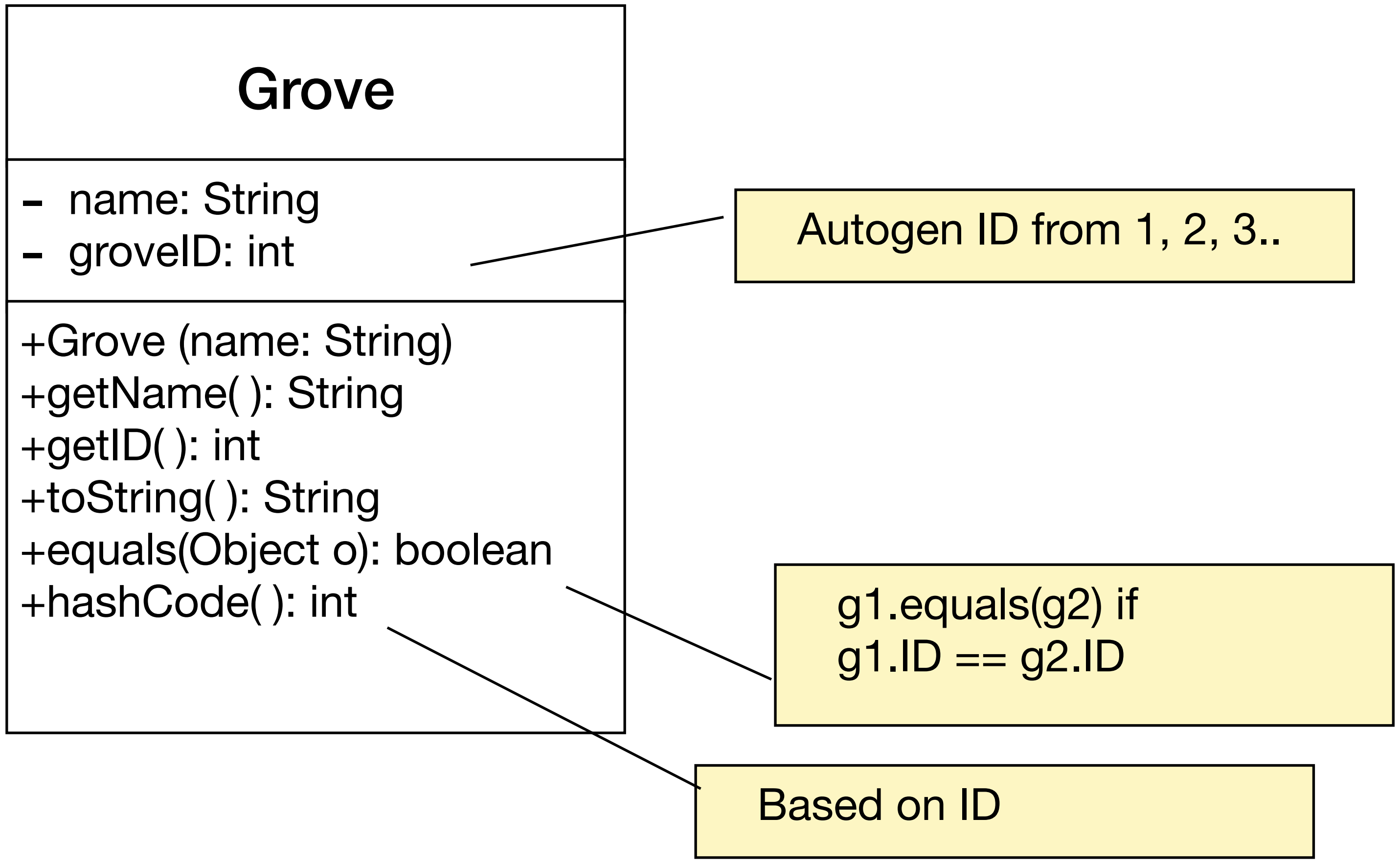


Valencia Orange

A-Z Sorting

Grove Class

Suppose we have a Grove class as follows:



Grove

- name: String
- groveID: int

+Grove (name: String)
+getName( ): String
+getID( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int

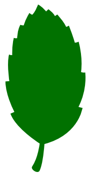
Autogen ID from 1, 2, 3..

g1.equals(g2) if
g1.ID == g2.ID

Based on ID

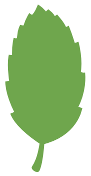
Grove Class

A Grove represents a collection of specific types of plants



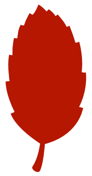
Pineapple Orange

(Grove)



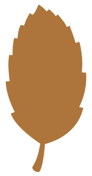
Blood Orange

(Grove)



Navel Orange

(Grove)



Valencia Orange

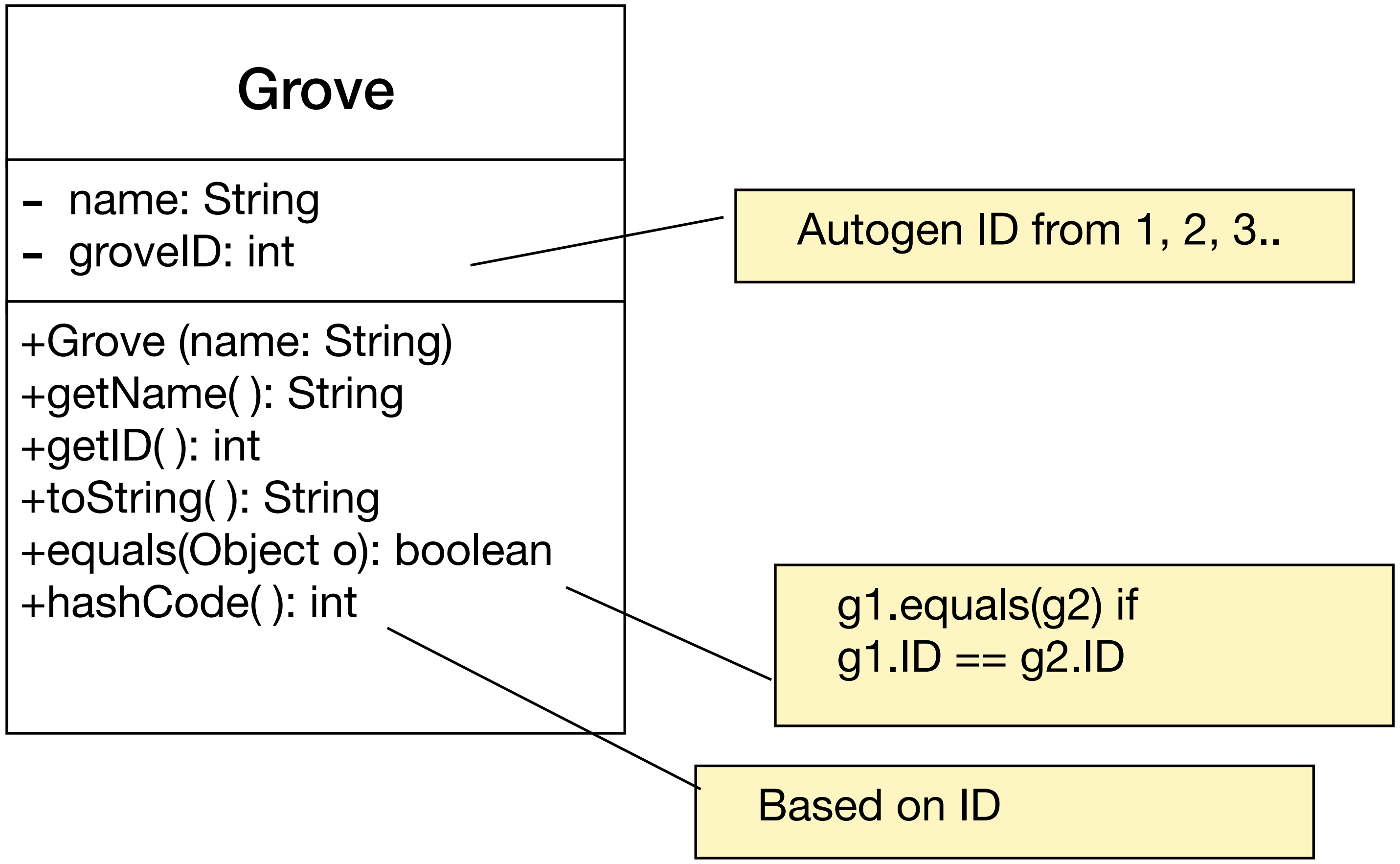
(Grove)

Tangerine

(Grove)

Clementine

(Grove)



Grove

- name: String
- groveID: int

+Grove (name: String)
+getName( ): String
+getID( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int

Autogen ID from 1, 2, 3..

g1.equals(g2) if
g1.ID == g2.ID

Based on ID

Grove Class

A Grove represents a collection of specific types of plants

Blood Orange

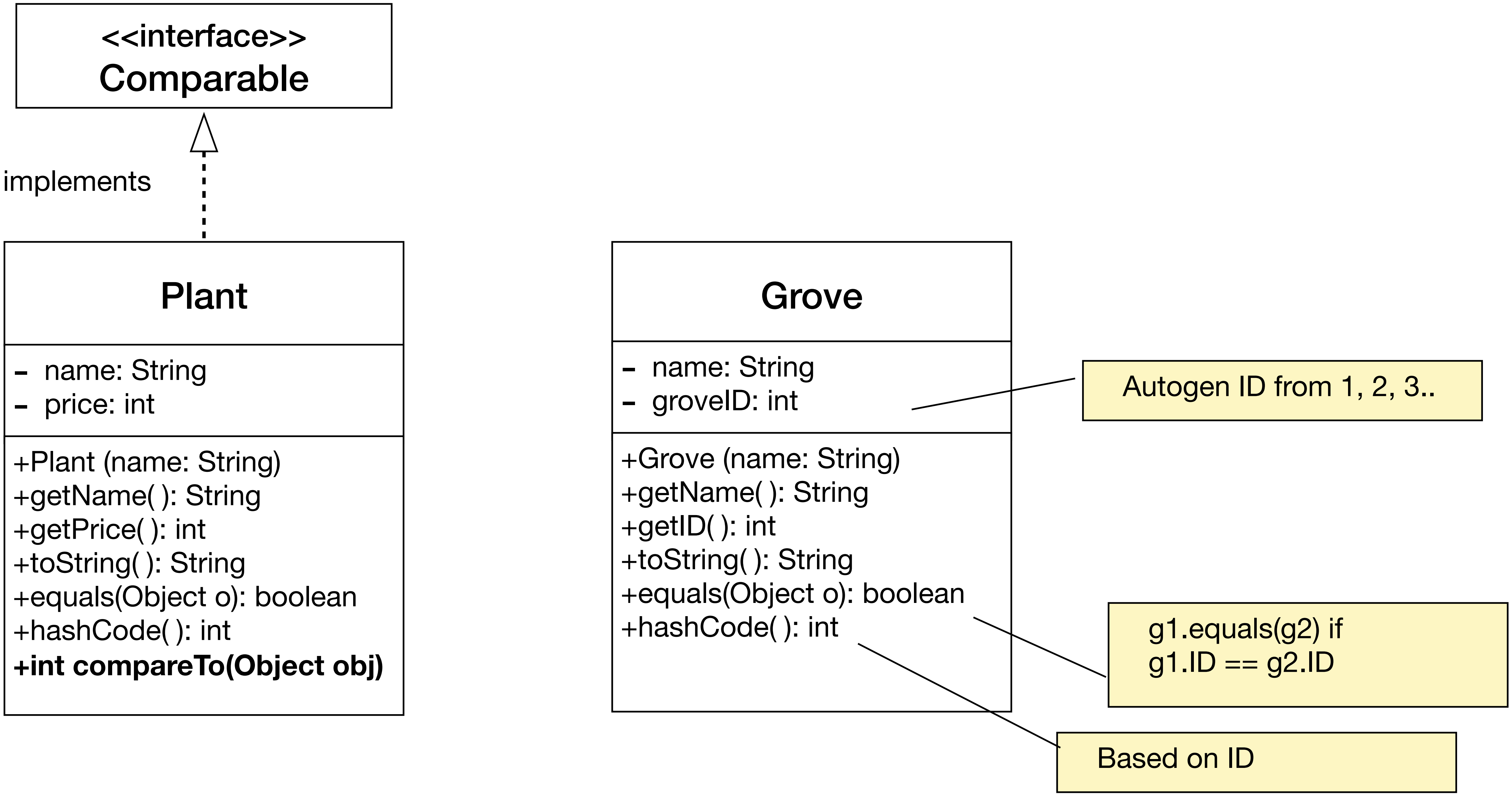
Clementine

Navel Orange

Pineapple Orange

Tangerine

Valencia Orange



<<interface>>
Comparable

implements

Plant

- name: String
- price: int

+Plant (name: String)
+getName( ): String
+getPrice( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int
+int compareTo(Object obj)

Grove

- name: String
- groveID: int

+Grove (name: String)
+getName( ): String
+getID( ): int
+toString( ): String
+equals(Object o): boolean
+hashCode( ): int

Autogen ID from 1, 2, 3..

g1.equals(g2) if
g1.ID == g2.ID

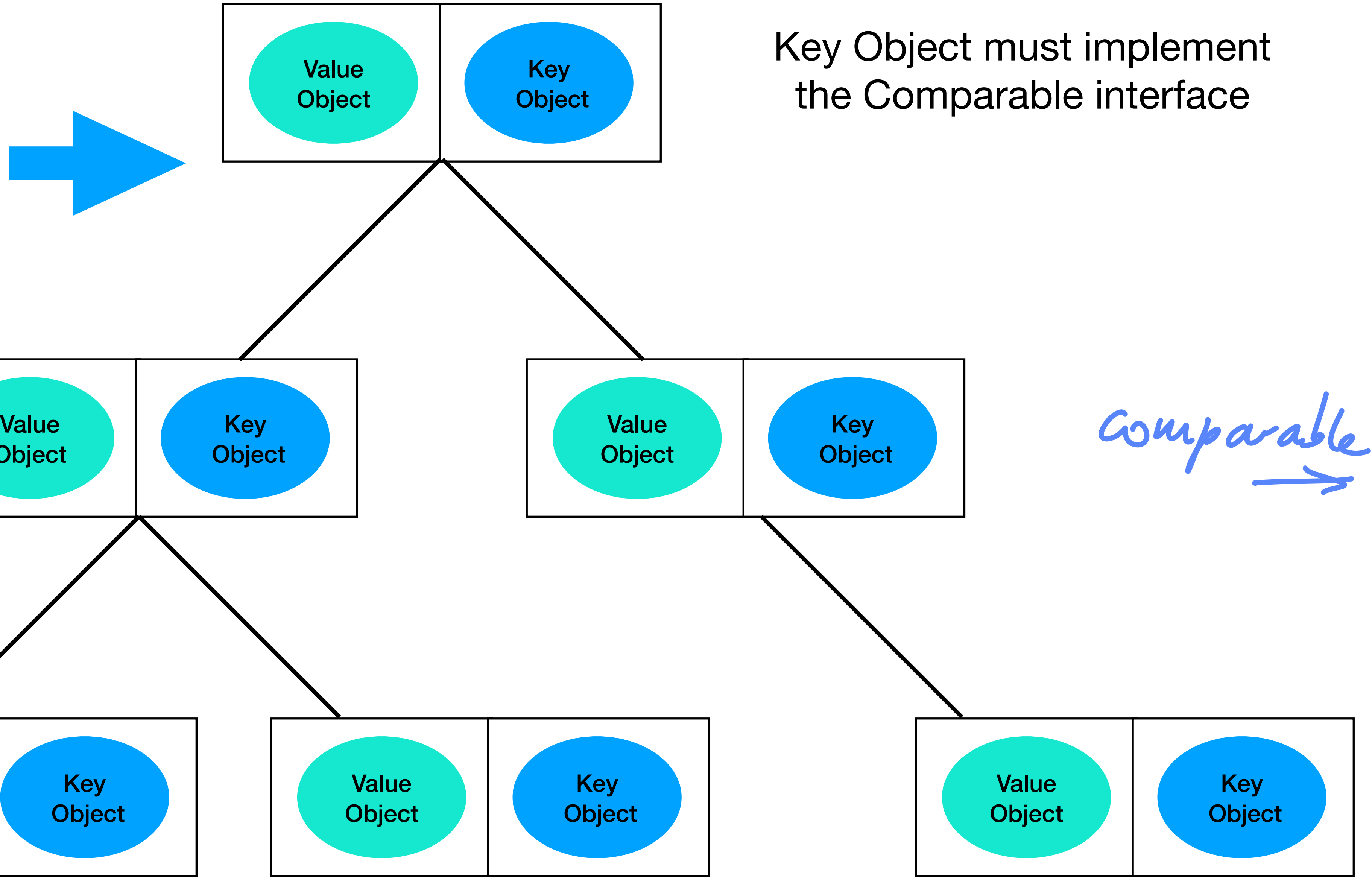
Based on ID

Storage in a TreeMap

Mangrove grove

Suppose we wish to store Grove and Plant objects in a TreeMap

buffalo



Key Object must implement the Comparable interface

comparable

Key (Plant)

Value (Grove)

TreeMap

Creating and instantiating with custom Object key:

Key type **Value type**

```
TreeMap <Plant,Grove> farm; // declaration
farm = new TreeMap<>(); // initialisation
```

The Plant class must implement the Comparable interface.

The Plant's compareTo( ) is used to sort the Plant objects (keys)

https://guide.michelin.com/en/article/features/orange-guide-fruit-navel-blood

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

TreeMap

Creating and instantiating with custom Object key, and Comparator:

Key type **Value type**

```
TreeMap <Plant,Grove> farm; // declaration
PlantComparator comparator = new PlantComparator();
farm = new TreeMap<>(comparator); // initialisation
```

The PlantComparator's compare( ) will be used to sort the Plant objects (keys)

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

TreeMap - Adding data

Adding objects (key, value pair) - custom key :

```
String groveName = p1.getName();
//Pineapple Orange
Grove g100 = new Grove(groveName);
//created Grove with supplied name
farm.put(p1,g100);
//inserted into TreeMap. Plant is Comparable
```

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

TreeMap - Adding data

Adding objects (key, value pair) - custom key :

```
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");
farm.put(p2, new Grove(p2.getName()));
farm.put(p3, new Grove(p3.getName()));
farm.put(p4, new Grove(p4.getName()));
farm.put(p5, new Grove(p5.getName()));
farm.put(p6, new Grove(p6.getName()));
```

p1

(k, v)

6 keys   6 values

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

Runner Class

```
import java.util.*;
public class Runner{
     public static void main(String[] args){
      TreeMap<Plant,Grove> farm = new TreeMap<Plant,Grove>();

            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"); //add (key,value) pair
            String groveName = p1.getName(); //returns Pineapple Orange
            Grove g100 = new Grove(groveName); //creates Grove with supplied name
            farm.put(p1,g100); //inserting into TreeMap. Plant is Comparable

            farm.put(p2, new Grove(p2.getName()));
            farm.put(p3, new Grove(p3.getName()));
            farm.put(p4, new Grove(p4.getName()));
            farm.put(p5, new Grove(p5.getName()));
            farm.put(p6, new Grove(p6.getName()));

     }
}
```

Runner Class - TreeMap state

.values()

```
MAP:
Plant: Blood Orange          $49 => Grove 200 Blood Orange
Plant: Clementine            $55 => Grove 600 Clementine
Plant: Navel Orange          $64 => Grove 300 Navel Orange
Plant: Pineapple Orange  $62 => Grove 100 Pineapple Orange
Plant: Tangerine             $56 => Grove 500 Tangerine
Plant: Valencia Orange      $14 => Grove 400 Valencia Orange
```

.keySet()

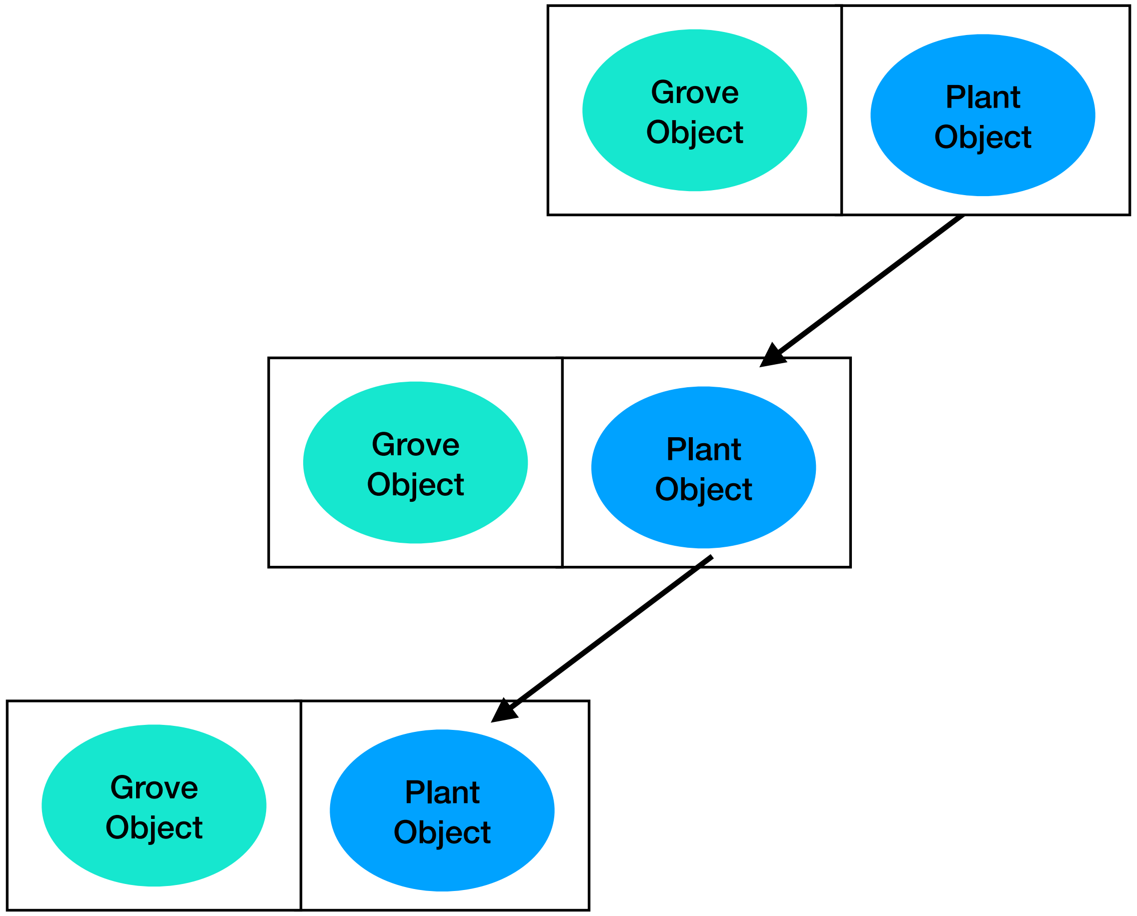
TreeMap - Getting value

Getting an object (value) - custom Key

```
   Plant key = new Plant("Blood Orange”);

   /* retrieves the Grove object that is mapped to the key
   using the compareTo( ) method of the Plant class to locate the
   value*/
1. Grove value = farm.get(key);

   System.out.println(value);
```



Output >> Grove 200 Blood Orange

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

TreeMap - Getting all values

Getting all objects (values): returns the values in ascending order of the corresponding keys.

```
Collection<Grove> groveObjects = farm.values();
System.out.println(groveObjects);
```

Output

>> [Grove 200 Blood Orange, Grove 600 Clementine, Grove 300 Navel Orange, Grove 100 Pineapple Orange, Grove 500 Tangerine, Grove 400 Valencia Orange]

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

TreeMap - Getting all keys

Getting all objects (keys): returns the keys in ascending order.

```
Set<Plant> plantObjects = plants.keySet();
System.out.println(plantObjects);
```

Output

```
 >> [Plant: Blood Orange      $90, Plant: Clementine   $21, Plant:
Navel Orange     $29, Plant: Pineapple Orange      $21, Plant:
Tangerine     $1, Plant: Valencia Orange    $24]
```

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

TreeMap

Testing for an object (key): Returns `true` if the map contains a mapping for the specified key.

```
boolean hasKey = plants.containsKey(“Sunflower”);
```

Testing for an object (value): Returns `true` if the map maps one or more keys to the specified value.

```
Plant p = new Plant(“Ixora”);
boolean hasKey = plants.containsValue(p);
```

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

TreeMap - print all data

In order to print all key, value pairs, first the collection of keys are retrieved, next the collection is traversed and the value mapped to each key is found, and the pair printed.

```
1. Set<Plant> plants = farm.keySet(); //get keys
2. java.util.Iterator<Plant> iter = plants.iterator();
3. while(iter.hasNext()){
4.     Plant plant = iter.next();
5.     Grove grove = map.get(plant); //get value mapped to key
       System.out.println(plant + "\t=> " + grove);
   }
```

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

TreeMap - find a key given the value

In order to find a key mapped to a value, first the collection of values are retrieved, next the collection is traversed and the value mapped to each key is checked for equality against the supplied value, and if found then the key is returned.

```
   // Locate the key for Grove object g100
1. Plant grove100Key = null;  // key will be a Plant object if found
2. Iterator<Plant> iter = farm.keyset().iterator(); //all keys
3. while(iter.hasNext() && grove100Key == null){
    4. Plant key = iter.next();
 → 5. if(farm.get(key).equals(g100)){// is this key mapped to g100?
          grove100Key = key; break;
   }    }
 → System.out.println(grove100Key);
```

Example - Plant Class

(If used as the key to a TreeMap)

value in TreeSet

```
public class Plant implements Comparable{
    private String name;

    public boolean equals(Object obj){…}
    …
    // Compare by Name – ascending A-Z
    public int compareTo(Object obj){
       if(obj instanceof Plant){
         Plant p = (Plant)obj;
          return this.name.compareTo(p.name);
       }
       throw new ClassCastException(“Not a Plant”);
    }
}
```

Exercise

Homework

1) Declare and instantiate a TreeMap, called **plants**, that stores a Plant object using the plant’s name as the key

2) Declare and instantiate a second polymorphic, called **herbs**, TreeMap that stores a Plant object using the plant’s name as the key

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