Container Classes

Sets, Sorted Sets

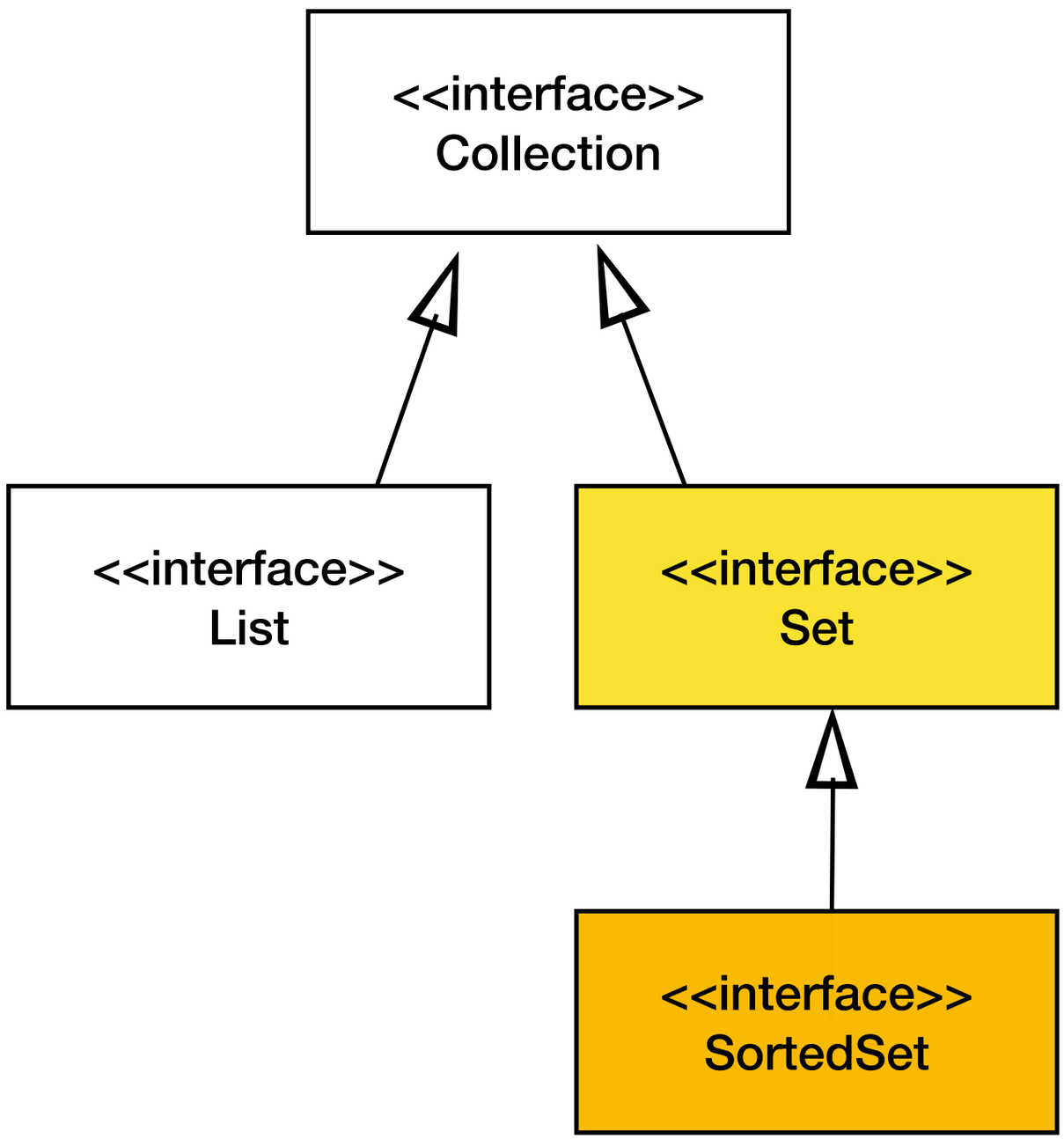
COMP2603
Object Oriented Programming 1

Week 8, Lecture 2

Outline

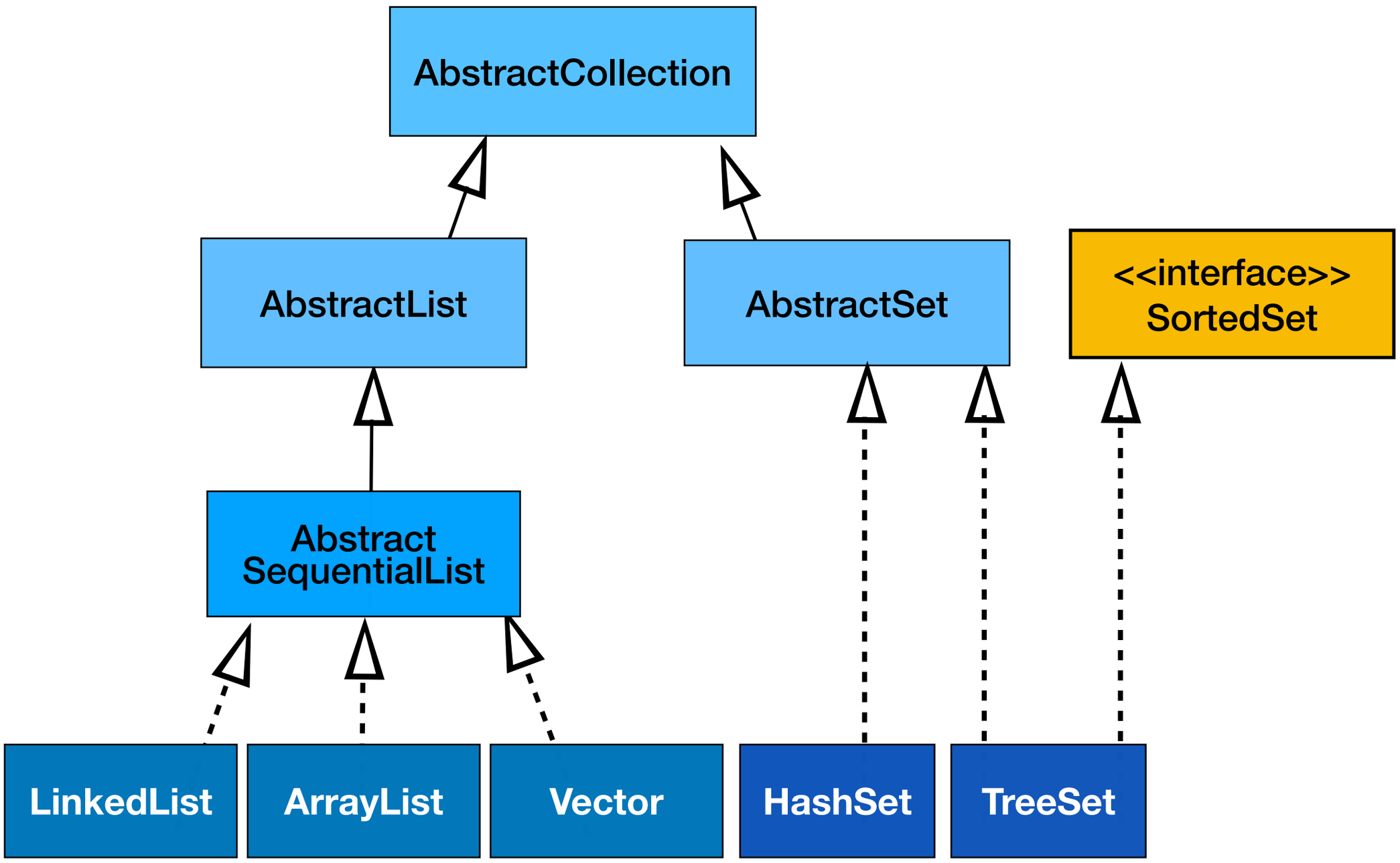
- Java Collections Framework
- Collection Interface:
    - Linked List ✓
    - ArrayList ✓
    - Vector ✓
  - Set: HashSet
    - SortedSet: TreeSet
- Comparable Interface
- Comparator Interface

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 Set Interface

- The Set interface represents an unordered collection of objects that contains no duplicate elements.
- It therefore cannot contain two elements e1 and e2 where e1.equals(e2).
- A Set can contain at most one null element.
- The Set interface declares the same methods as its super-interface, Collection.

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

The Set Interface
(same as Collection)

Method	Description
boolean add (E o)	Inserts the object of the specified type into the collection; returns true if the object was added, false otherwise
boolean addAll (Collection c)	Inserts all the objects from the specified collection into the current collection
void clear( )	Removes all the elements from the collection
boolean contains (Object o)	Returns true id the specified object is present in the colleciton, and false otherwise
boolean isEmpty( )	Returns true if there are no elements in the collection, and false otherwise
boolean remove(Object o)	Deletes the specified object from the collection
int size( )	Returns the number of elements currently in the collection

The Set Interface - Adding Elements

The Set interface does not allow the add( ) and addAll( ) methods to duplicate elements to the Set.

If a Set already contains the element being added, its add( ) and addAll( ) methods return false.

In order for a Set to determine if it already contains an element, it uses the equals( ) method of the element to check if the element is equal to an existing element in the Set.

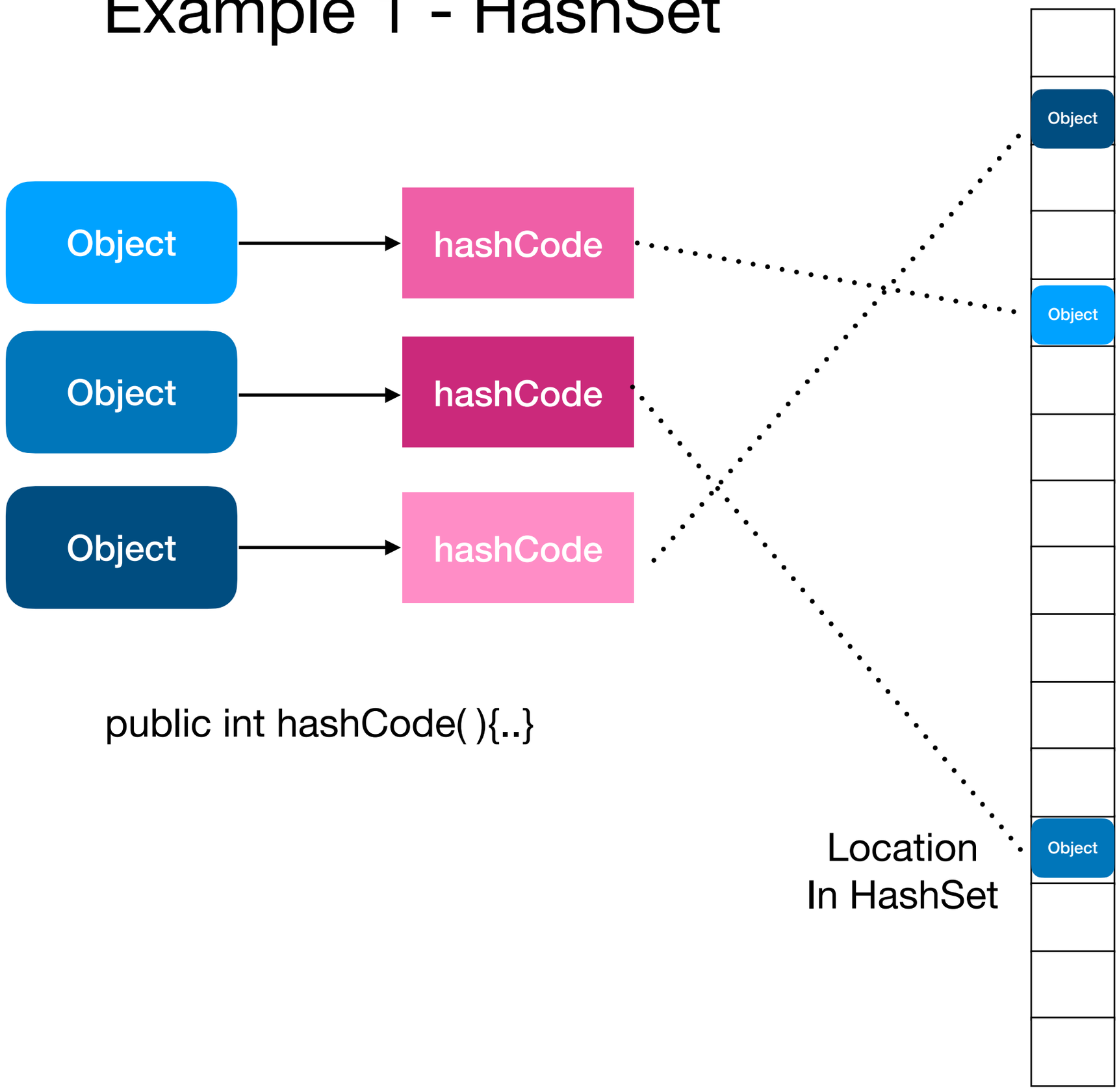
It is therefore necessary to override the equals( ) method of the objects that will be inserted into a Set.

HashSet

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.

Example 1 - HashSet



public int hashCode( ){..}

Example 1 - HashSet

Demo - W8 demo L2

Demo

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

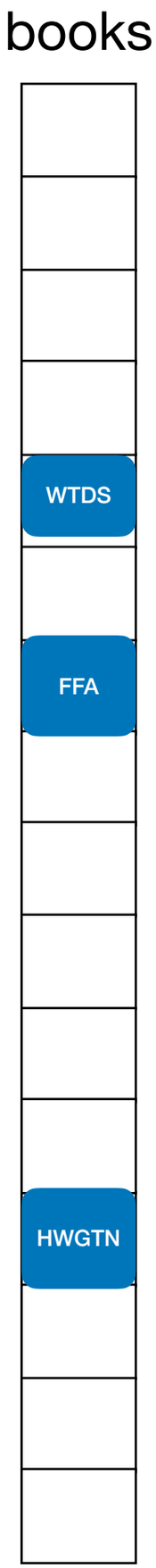
```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        HashSet<String> books = new HashSet<>();

        String b1 = "What the dog saw";
        String b2 = "Flowers for Algernon";
        String b3 = "How we got to now";

        books.add(b1);
        books.add(b2);
        books.add(b3);

        System.out.println(books);


    }

}
```

Example 1 - HashSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, What the dog saw, How we got to now]
```

“Flowers for Algernon”.hashCode( ) -> “Flowers for Algernon”



Example 1 - HashSet

Demo - W8 demo L2

Demo

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

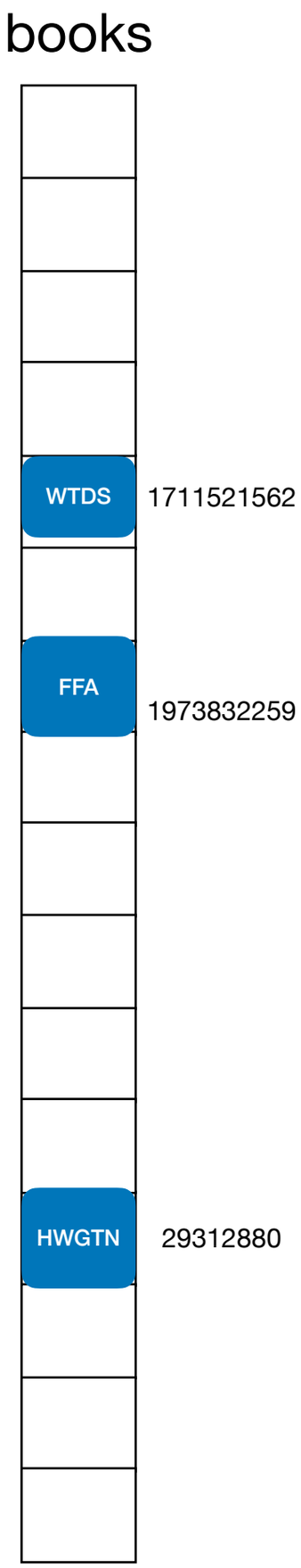
```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        HashSet<String> books = new HashSet<>();

        String b1 = "What the dog saw";
        String b2 = "Flowers for Algernon";
        String b3 = "How we got to now";

        books.add(b1);
        books.add(b2);
        books.add(b3);

        System.out.println(books);
        System.out.println("["+b2.hashCode()+","
                           +b1.hashCode()+","
                           +b3.hashCode()+"]");

    }

}
```

Example 1 - HashSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, What the dog saw, How we got to now]
[1973832259,1711521562,293128807]
```

“Flowers for Algernon”.hashCode( ) -> “Flowers for Algernon” (1973832259)



Example 2 - HashSet

Plant - W8 demo L2

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Plant{
    private String name;
    public Plant(String name){
        this.name = name;
    }
    public String toString(){
        return name;
    }
    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");
    }
}
```

Example 2 - HashSet

Demo - W8 demo L2

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Basil");
        Plant p3 = new Plant("Basil");
        HashSet<Plant> herbs = new HashSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);

    }
```

Example 2 - HashSet

Demo - W8 demo L2

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Basil");
        Plant p3 = new Plant("Basil");
        HashSet<Plant> herbs = new HashSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);

    }

```

BlueJ: Terminal Window - W8 demo L2

```
[Basil, Basil, Basil]
```

Example 2 - HashSet

Demo - W8 demo L2

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Basil");
        Plant p3 = new Plant("Basil");
        HashSet<Plant> herbs = new HashSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);
        System.out.println("["+p1.hashCode()+","
                           +p2.hashCode()+","
                           +p3.hashCode()+"]");

    }
```

BlueJ: Terminal Window - W8 demo L2

```
[Basil, Basil, Basil]
[806172960,938479544,759002479]
```

Example 2 - HashSet

Demo - W8 demo L2

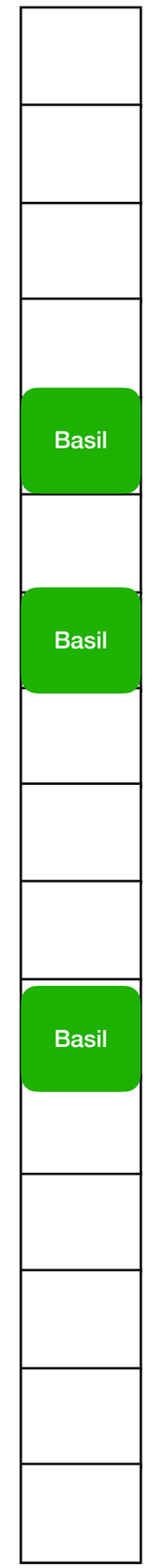
Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Basil");
        Plant p3 = new Plant("Basil");
        HashSet<Plant> herbs = new HashSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);
        System.out.println("["+p1.hashCode()+","
                           +p2.hashCode()+","
                           +p3.hashCode()+"]");

    }
```

BlueJ: Terminal Window - W8 demo L2

```
[Basil, Basil, Basil]
[806172960,938479544,759002479]
```



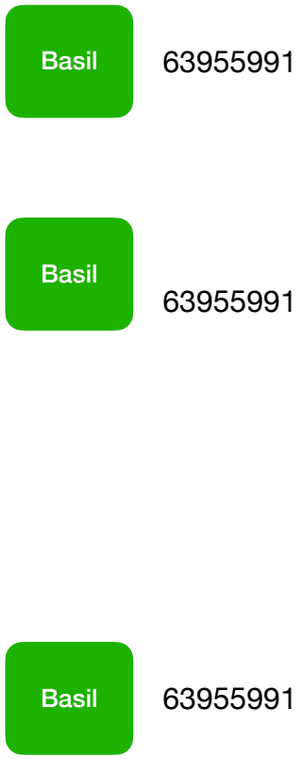
Example 2 - HashSet

Plant - W8 demo L2

Demo × Plant ×

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
public class Plant{
    private String name;
    public Plant(String name){
        this.name = name;
    }
    public String toString(){
        return name;
    }
    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();
    }
}
```



Example 2 - HashSet

Demo - W8 demo L2

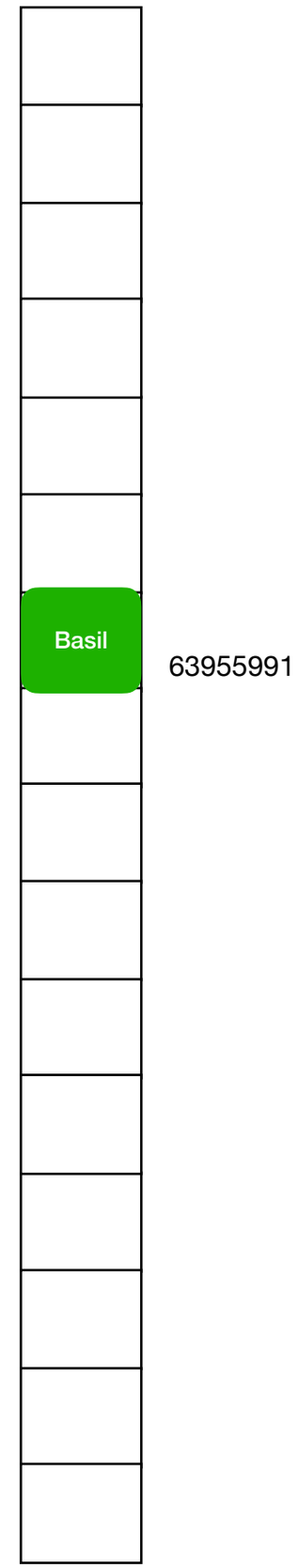
Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;

public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Basil");
        Plant p3 = new Plant("Basil");
        HashSet<Plant> herbs = new HashSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);
        System.out.println("["+p1.hashCode()+","
                           +p2.hashCode()+","
                           +p3.hashCode()+"]");

    }
```

BlueJ: Terminal Window - W8 demo L2

```
[Basil]
[63955991,63955991,63955991]
```



Example 3 - TreeSet

Demo - W8 demo L2

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;
import java.util.TreeSet;
public class Demo{
    public static void main(String[] args){
        TreeSet<String> books = new TreeSet<>();

        String b1 = "What the dog saw";
        String b2 = "Flowers for Algernon";
        String b3 = "How we got to now";

        books.add(b1);
        books.add(b2);
        books.add(b3);

        System.out.println(books);
    }
```

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, How we got to now, What the dog saw]
```

Example 3 - TreeSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, How we got to now, What the dog saw]
```

Example 1 - HashSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, What the dog saw, How we got to now]
```

TreeSet

A NavigableSet implementation based on a TreeMap. The elements are ordered using their **natural ordering**, or by a **Comparator** provided at set creation time, depending on which constructor is used

The Sorted Set Interface

The SortedSet interface is a Set that sorts its elements and guarantees that elements are enumerated in sorted order.

It declares a few methods of its own such as first( ) and last( ) which return the lowest and highest elements in the set, respectively (as determined by the sort order).

String

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP **Java SE 21 & JDK 21**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH Search

Module java.base
Package java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc

---

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
```

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

String

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP **Java SE 21 & JDK 21**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH Search

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

For finer-grained String comparison, refer to `Collator`.

Specified by:

`compareTo` in interface `Comparable<String>`

Parameters:

`anotherString` - the `String` to be compared.

Returns:

the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

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

Example 3 - TreeSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, How we got to now, What the dog saw]
```

Example 1 - HashSet

BlueJ: Terminal Window - W8 demo L2

```
[Flowers for Algernon, What the dog saw, How we got to now]
```

Example 4 - TreeSet

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;
import java.util.TreeSet;
public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil");
        Plant p2 = new Plant("Anise");
        Plant p3 = new Plant("Thyme");
        TreeSet<Plant> herbs = new TreeSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);

    }
```

java.lang.ClassCastException:
class Plant cannot be cast to class java.lang.Comparable (Plant is in unnamed module of loader java.net.URLClassLoader @5a31df4e; java.lang.Comparable is in module java.base of loader 'bootstrap') (in java.util.TreeMap)

saved

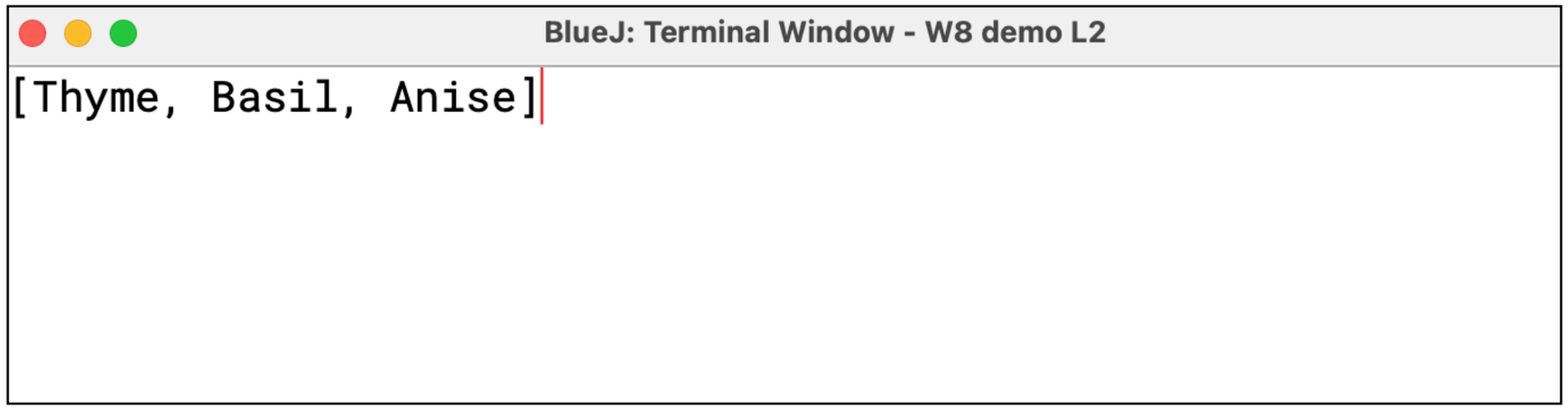
Example 4 - TreeSet

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Plant implements Comparable{
    private String name;
    public Plant(String name){
        this.name = name;
    }
    public String toString(){
        return name;
    }
    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 p.name.compareTo(this.name);
        }
        throw new ClassCastException(
        "Object must be a plant for comparison");
    }
}
```

Example 4 - TreeSet



The plants are now sorted in reverse alphabetical order.

Example 4 - TreeSet

The plants are now sorted in reverse alphabetical order.

BlueJ: Terminal Window - W8 demo L2

```
[Thyme, Basil, Anise]
```

To sort the plants in alphabetical order:

```
20      public int compareTo(Object obj){
21          if (obj instanceof Plant){
22              Plant p = (Plant) obj;
23              return this.name.compareTo(p.name);
24          }
25          throw new ClassCastException(
26          "Object must be a plant for comparison");
27      }
28  }
```

BlueJ: Terminal Window - W8 demo L2

```
[Anise, Basil, Thyme]
```

Comparator Interface

A comparison function, which imposes a total ordering on some collection of objects.

Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order.

Comparators can also be used to control the order of certain data structures (such as sorted sets or sorted maps), or to provide an ordering for collections of objects that don't have a natural ordering.

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

Example 5 - TreeSet & Comparator

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
public class Plant implements Comparable{
    private String name;
    private int cost;
    public Plant(String name){
        this.name = name;
    }
    public Plant(String name, int cost){
        this.name = name;
        this.cost = cost;
    }
    public int getCost(){return cost;}
    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 5 - TreeSet & Comparator

Demo × Plant ×

Compile Undo Cut Copy Paste Find... Close Source Code

```java
import java.util.HashSet;
import java.util.TreeSet;
public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil",20);
        Plant p2 = new Plant("Anise",25);
        Plant p3 = new Plant("Thyme",14);
        TreeSet<Plant> herbs = new TreeSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);
}
```

BlueJ: Terminal Window - W8 demo L2

```
[Anise $25, Basil $20, Thyme $14]
```

Example 5 - TreeSet & Comparator

Demo × | Plant × | PriceComparator ×

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
import java.util.Comparator;
public class PriceComparator implements Comparator{
    public int compare(Object obj1, Object obj2){
        if(obj1 instanceof Plant && obj2 instanceof Plant){
            Plant p1 = (Plant)obj1;
            Plant p2 = (Plant)obj2;
            if (p1.getCost() == p2.getCost()) return 0;
            if (p1.getCost() > p2.getCost()) return 1;
            if (p1.getCost() < p2.getCost()) return -1;
        }
        throw new ClassCastException(
        "Objects must be plants for comparison");
    }
}
```

Example 5 - TreeSet & Comparator

Demo × | Plant × | PriceComparator ×

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
import java.util.HashSet;
import java.util.TreeSet;
public class Demo{
    public static void main(String[] args){
        Plant p1 = new Plant("Basil",20);
        Plant p2 = new Plant("Anise",25);
        Plant p3 = new Plant("Thyme",14);
        TreeSet<Plant> herbs = new TreeSet<>();
        herbs.add(p1);
        herbs.add(p2);
        herbs.add(p3);
        System.out.println(herbs);

        PriceComparator priceComp = new PriceComparator();
        TreeSet<Plant> herbsByCost = new TreeSet<>(priceComp);
        herbsByCost.addAll(herbs);
        System.out.println(herbsByCost);
    }
```

BlueJ: Terminal Window - W8 demo L2

```
[Anise $25, Basil $20, Thyme $14]
[Thyme $14, Basil $20, Anise $25]
```

Practice Questions

1. Write a Java method to remove **duplicates** from an ArrayList using a HashSet.
2. Write a Java method to find the **union** of two sets.
3. Write a Java method to check if a set is a **subset** of another set.
4. Write a Java method to find the **intersection** of two sets.

   Tip: Look at the Set API for methods that end in ‘all’ then use appropriately.

Summary

Today you learned about:

- Concrete Sets: HashSet, TreeSet
- The importance of the hashCode( ) method
  - Creation of a custom hashCode( )
- Sorted Collections: TreeSet
- Comparable Interface (used for Sorted Collections)
  - Implementation of int compareTo(Object obj)
- Comparator Interface (used to sort collections)
  - Implementation of int compare(Object obj1, Object obj2)