

Container Classes

Collection, List, Set

COMP2603
Object Oriented Programming 1

Week 8

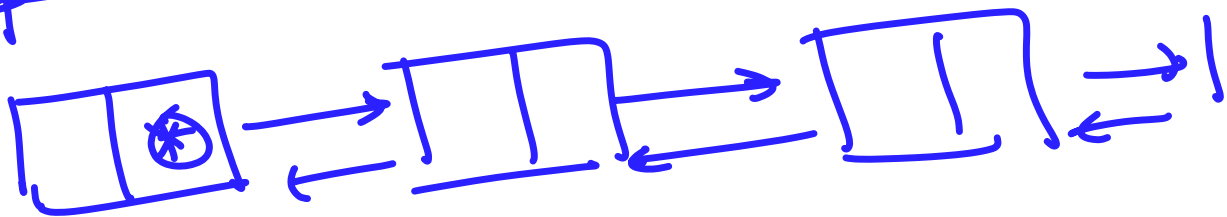
Outline

- Java Collections Framework
- Collection Interface
- List
 - Linked List
- Set
 - SortedSet

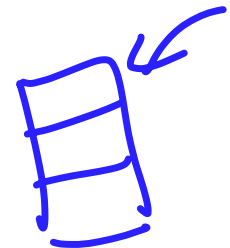
Application Programming Interface (API)

What is an Application Programming Interface (API)?

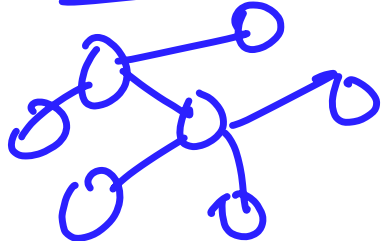
HashMap 

Linked List 

Binary Tree 

Stack 

Queue 

Graphs 

Arrays 

<https://www.youtube.com/watch?v=s7wmiS2mSXY>

Java Collections Framework

The **Java Collections** framework contains a set of interfaces such as:

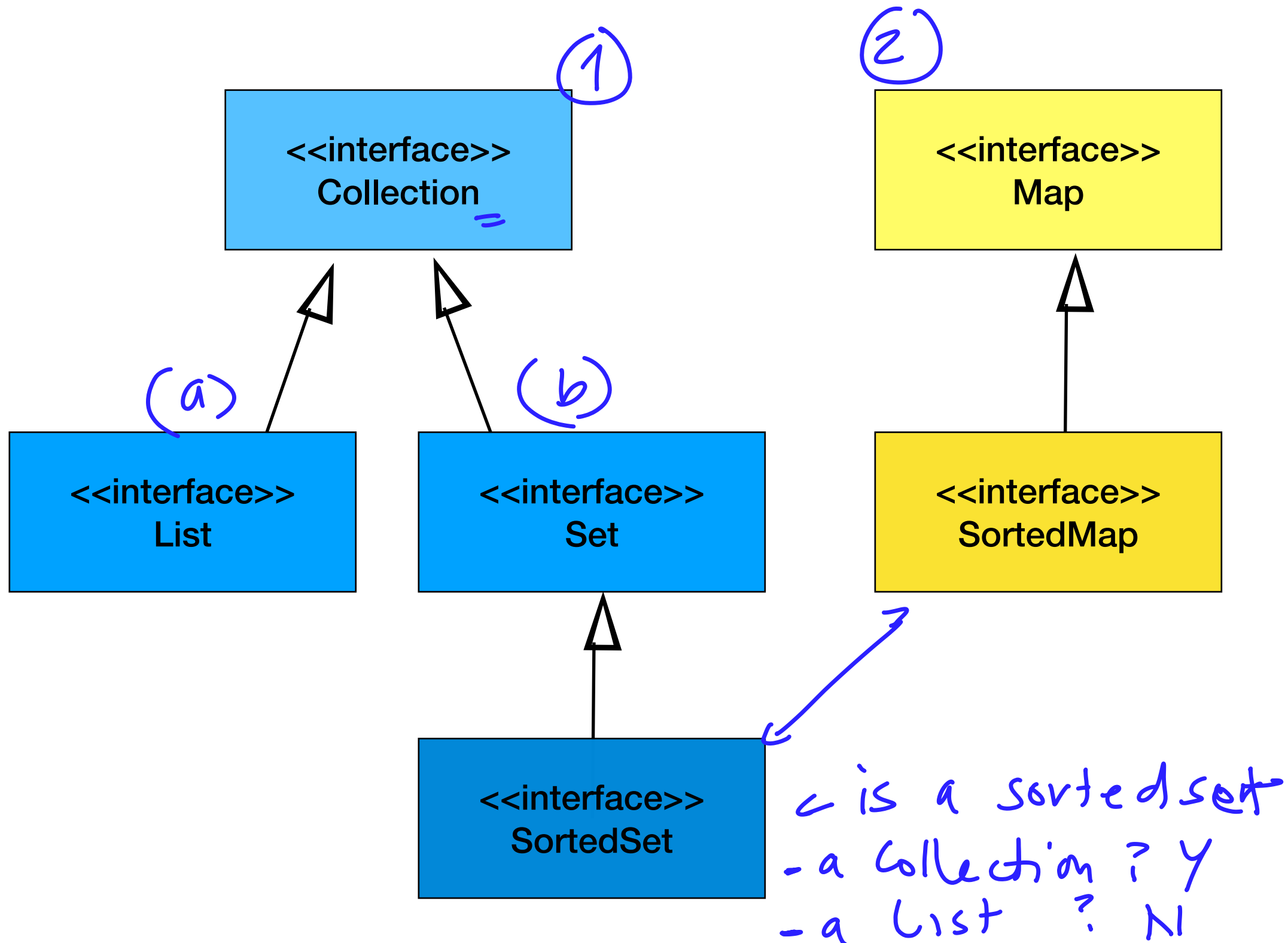
- **Collection**
- **Set**
- **List**
- **Map.**

These are used as the basis for creating concrete collection classes such as **ArrayList**, **LinkedList**, **HashSet**, and **TreeMap**.

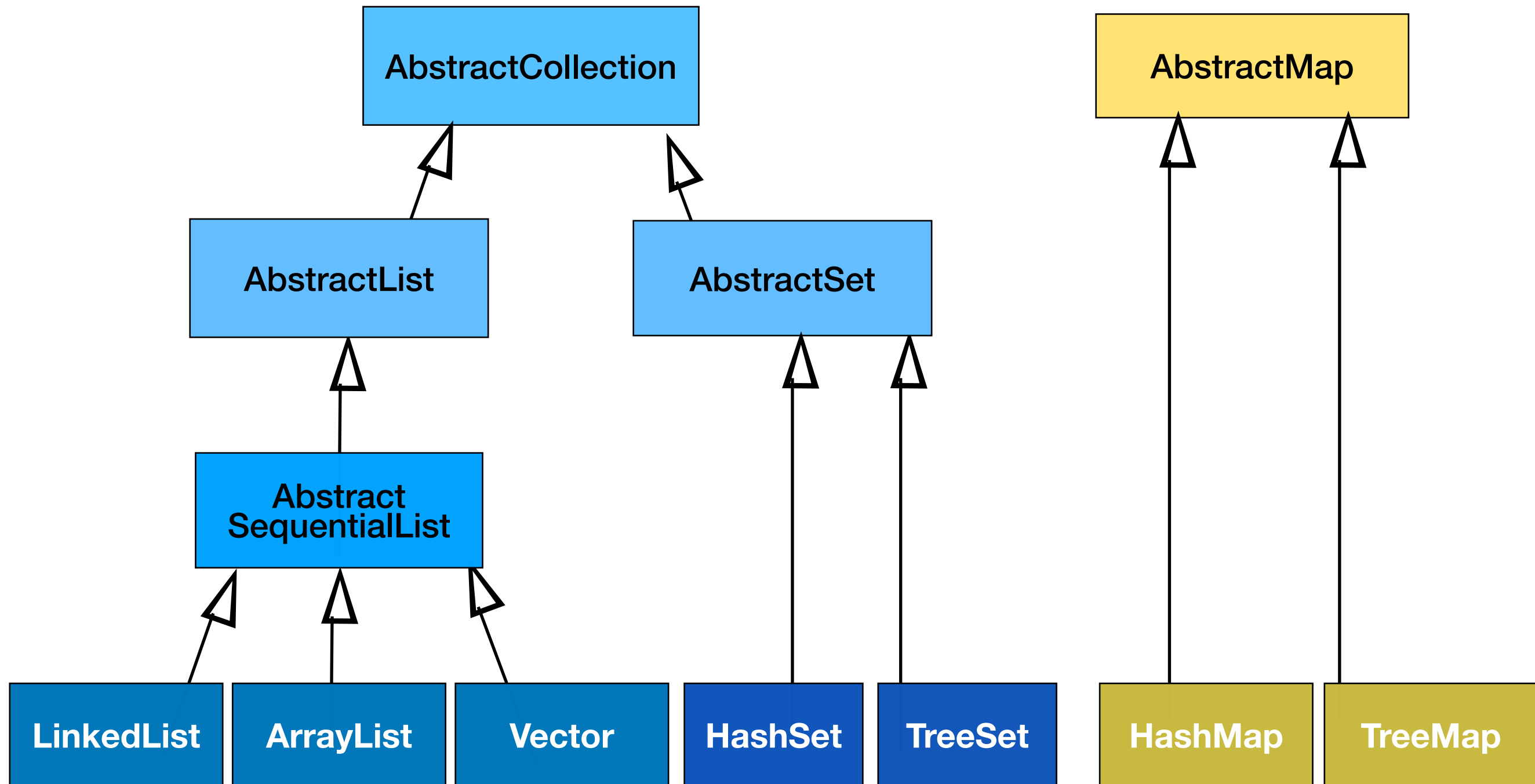
*

→

Interfaces in the Java Collections Framework



Classes in the Java Collections Framework



The Collection Interface

The **Collection** interface represents a group or collection of objects. The objects may or may not be ordered, and the collection may or may not contain duplicate objects.

The **Collection** interface is not usually implemented directly.

Most concrete collection classes implement one or more of the specific sub-interfaces.

Seq.1 (asc)
(des)

→ 12 45 6 103 9 0 50

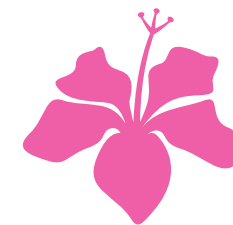
The Collection Interface

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

Simple Plant Class

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

-
1. `Plant p = new Plant("Hibiscus");`
 2. `System.out.println(p.toString());`



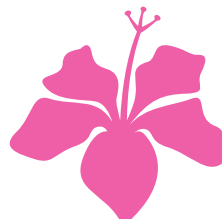
Outcome: Hibiscus

Simple Plant Class

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

```
Plant p = new Plant("Hibiscus");  
System.out.println(p); // Same as printing p.toString()
```

Outcome: Hibiscus



Example 0

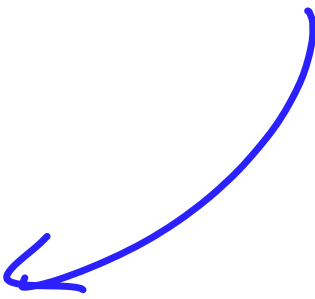

Collection<Plant> plants;

//Assume:

// (1) plants collection is initialised properly

// (2) plants collection filled with 5 Plant objects

1. int numPlants = plants.size();
2. System.out.println(numPlants);



Outcome: 5

Example 0.1

```
Collection<Plant> plants;
```

```
//Assume:
```

```
// (1) plants collection is initialised properly
```

```
// (2) plants collection filled with 5 Plant objects
```

```
1. int numPlants = plants.size();  
2. plants.clear();  
3. System.out.println(numPlants);
```

Outcome: 5

Example 0.2

```
Collection<Plant> plants;
```

```
//Assume:
```

```
// (1) plants collection is initialised properly
```

```
// (2) plants collection filled with 5 Plant objects
```

```
1. plants.clear();
```

```
2. System.out.println(plants.size());
```

num element

Outcome: 0

Example 0.3

1. ~~Collection~~^X<Plant> plants;

//plants is initialised, filled with 5 Plant objects

2. int numPlants = plants.size();

3. for(int i = 0; i < numPlants; i++){

~~Plant p = plants.get(i);~~ // fails..why?

System.out.println(p);

}

Outcome: X

Example 1

```
1. Collection<Plant> plants;
```

```
//Assume:
```

```
// (1) plants collection is initialised properly ✓
```

```
// (2) plants collection filled with 5 Plant objects
```

```
2. int numPlants = plants.size();
```

```
3. for(Plant p: plants){  
    → System.out.println(p);  
}
```

Outcome:

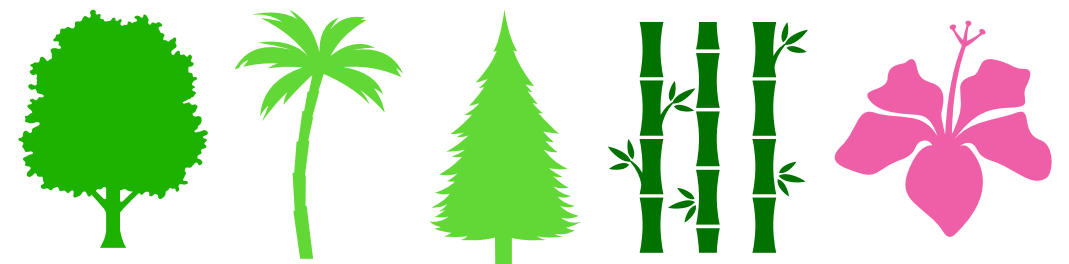
Poui

Coconut

Pine

Bamboo

Hibiscus



The Collection Interface

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

Example 2

equals() → loc

1. `Collection<Plant> plants;`

//plants is initialised and filled with Plant objects

2. `Plant hp = new Plant("Hibiscus");`  13

3. `if(plants.contains hp)){`

4 `plants.remove hp);`

}

x
equals()

≠

if (name.equals(p.name) → true

Outcome:

Poui

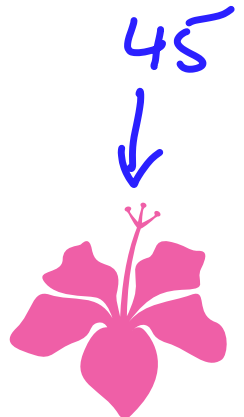
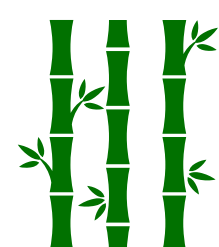
Coconut

Pine

Bamboo

Hibiscus

✓
<-Not removed..why?



Hibiscus

Example 2

```
Collection<Plant> plants;
//plants is initialised and filled with Plant objects
Plant hp = new Plant("Hibiscus");      //Plant@56a081b3
if(plants.contains(hp)){
    plants.remove(hp);
}
```

Outcome:

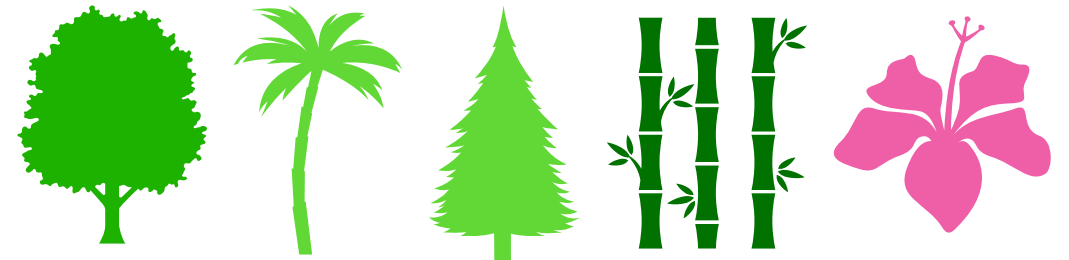
Poui

Coconut

Pine

Bamboo

Hibiscus <-Not removed..why?



Plant@36d3b4a2

Plant@26df8cb7

Plant@47b4f4f9

Plant@fbe288c

Plant@c401eb8

Example 2

```
public class Plant{
    private String name;

    public Plant(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
    public String toString(){
        return getName();
    }

    public boolean equals(Object obj){
        if (obj instanceof Plant){
            Plant p = (Plant)obj;
            return p.getName().equals(this.getName());
        }
        return false;
    }
}
```

Example 2

```
Collection<Plant> plants;
```

```
//plants is initialised and filled with Plant objects
```

```
Plant hp = new Plant("Hibiscus"); //Plant@56a081b3
```

```
{  
    if(plants.contains(hp)){  
        plants.remove(hp);  
    }  
}
```

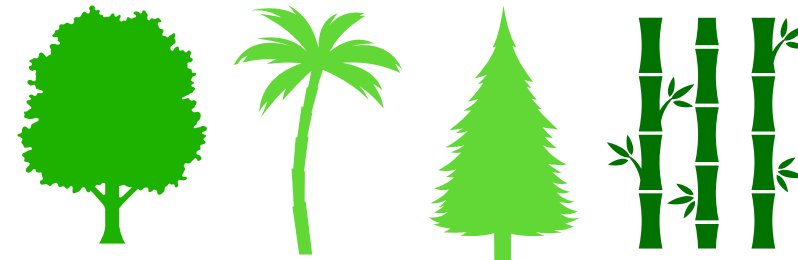
Outcome:

Poui

Coconut

Pine

Bamboo



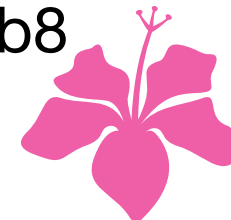
Plant@36d3b4a2

Plant@26df8cb7

Plant@47b4f4f9

Plant@fbe288c

Plant@c401eb8



Example 3

1. `Collection<Plant> plants; // = new TreeSet<>();`
 //plants is initialised and filled with Plant objects

2. `PineTree pt = new PineTree("Conifer");`

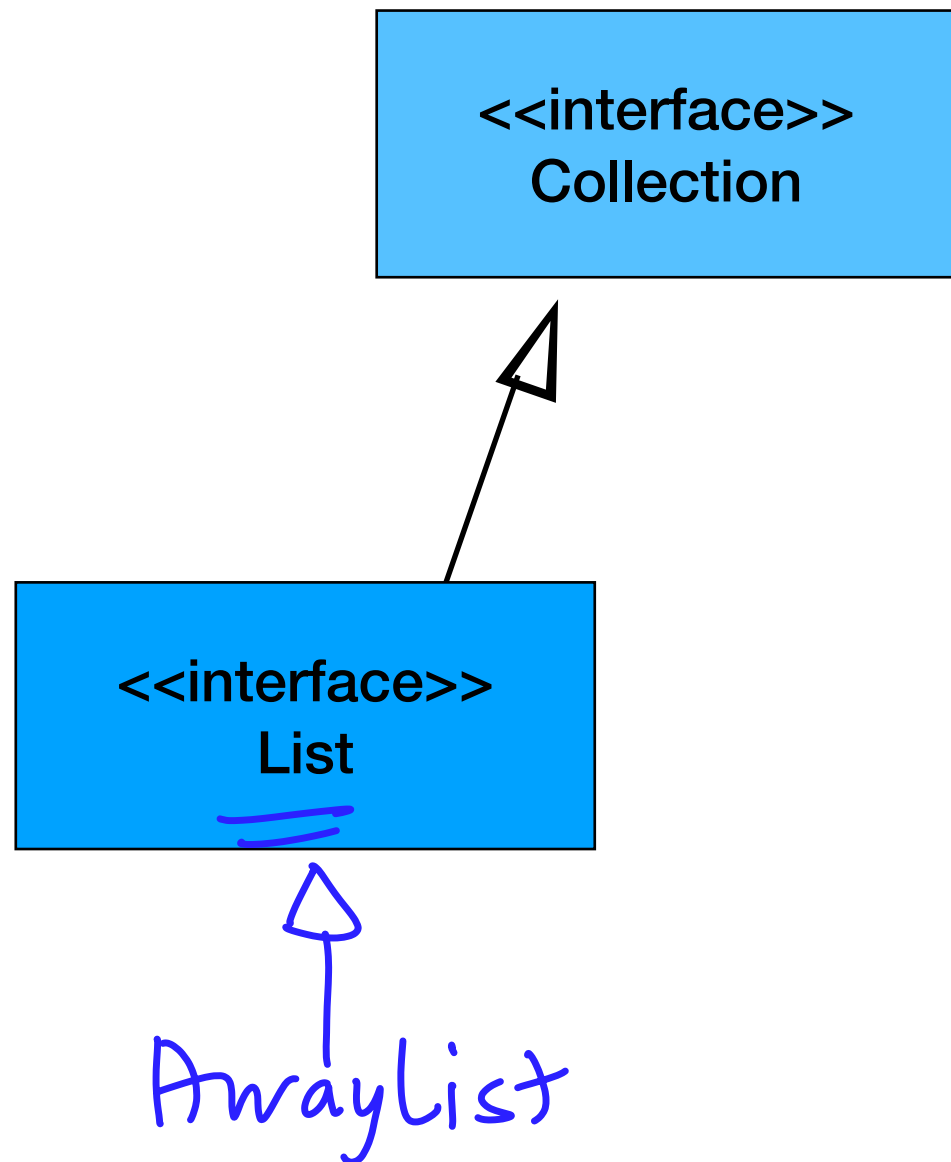
3. `plants.add(pt);` → F

Outcome:

X



Interfaces in the Java Collections Framework



The List Interface

The List interface represents an ordered collection of objects (also known as a sequence). Lists allow precise control over where each element is inserted.

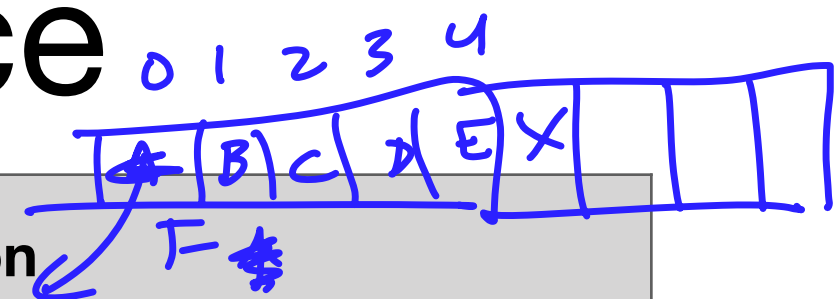
Each element in a List has an index, or position in the list, and elements can be inserted, queried, and removed by index.

1. The first element of a List has an index of zero.
2. The last element in a List has an index of size()-1.
3. Lists typically allow duplicate elements.

The List Interface

boolean add(E obj)

Overloaded



Method	Description
1. void <u>add</u> (int index, E <u>element</u>)	Inserts the object supplied as an argument in position index of the <u>List</u> . The other elements in the List are shifted down one position
2. E <u>get</u> (int index)	Returns the element at position index of the List
3. int <u>indexOf</u> (Object o)	Returns the position of the object supplied as an argument in the List, or -1 if it does not find a match. <u>indexOf()</u> uses the <u>equals()</u> method of the contained objects to check for equality with the object supplied as an argument.
4. E <u>remove</u> (int index)	Removes the object at position index of the List
5. E <u>set</u> (int index, E <u>element</u>)	Inserts the object supplied as element in position index of the List, overwriting the element that was there previously, if any. It returns the element that was previously stored at that position; otherwise it returns <u>null</u>

5/1

GO

Example 4

```
Collection<Plant> plants = new LinkedList<>();  
//plants is initialised, filled with 5 Plant objects  
int numPlants = plants.size();    //returns 5  
Plant p = plants.get(0); //fails  
  
if( plants instanceof LinkedList){  
    LinkedList<Plant> plantList = (LinkedList<>)plants;  
    Plant p = plantList.get(0);  
    System.out.println(p)  
}
```

plants collection:



Outcome:



Example 5

```
Collection<Plant> plants = new LinkedList<>();  
//plants is initialised, filled with 5 Plant objects  
int numPlants = plants.size();    //returns 5  
Plant p = plants.remove(0);  
    // compilation error- why?
```

plants collection:

