Object Oriented Development with Java

(CT038-3-2 and Version VC1)

Java Collection Framework

Java Collection APIs

Topic & Structure of The Lesson

- Collections
- Collection Interface
- Set Interface
- Hash set
- Linked Hash set
- Tree set
- List
- ArrayList
- -Vector

Learning outcome

- At the end of this lesson, you will be able to
  - Work with Java Collections Framework
  - Understand Generic type
  - Use `Collections` and `Arrays` classes

Key terms you must be able to use

If you have mastered this topic, you should be able to use the following terms correctly in your assessments:

- -Collections
- -Hash set
- -Tree set
- -List
- -ArrayList
- -Vector

Introduction

- Java API can organise and manipulate data efficiently through interfaces and classes.
- These interfaces and classes:
  - *Java Collections Framework (JCF)*.
- We will learn how to use these classes and interfaces in this lesson.

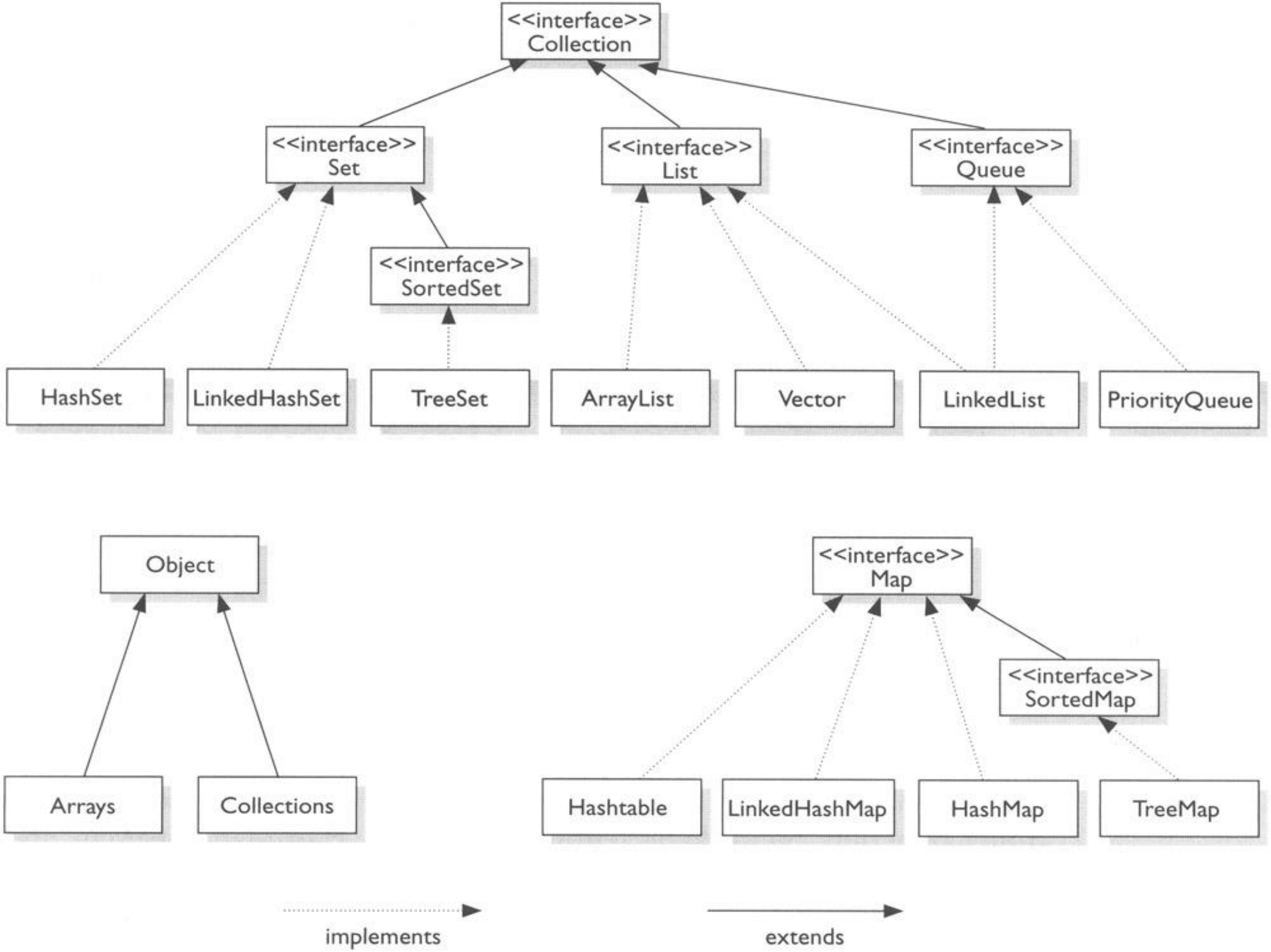
Collection

- A *collection* is a container object that stores a group of objects, often referred to as <u>elements</u>.
- JCF supports three types of collections: *set*, *list*, and *map*.
- They are made up by interfaces and classes. E.g., `Set` interface and its `HashSet` class.

Cont'd

- An instance of `Set` stores a group of nonduplicate elements.
- An instance of `List` stores an ordered collection of elements.
- An instance of `Map` stores a group of objects, each of which is associated with a key.

Note: Collection accepts only object reference type, not primitive type.


<<interface>> Collection
<<interface>> Set
<<interface>> List
<<interface>> Queue
<<interface>> SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
<<interface>> Map
<<interface>> SortedMap
Arrays
Collections
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

`Collection` interface

- Is the roof interface for manipulating a collection of objects.
- Provides basic operations for adding and removing elements in a collection.
  - The `add` method adds an element to the collection.
  - The `addAll` method adds all the elements in a specified collection to this collection.
  - And `remove` and `removeAll` methods.
  - More operations: http://docs.oracle.com/javase/7/docs/api/

`Set` interface

- `Set` interface extends `Collection` interface.
- The concrete classes that implement `Set` must ensure that <u>no duplicate elements</u> can be added to the set.
- That is no two elements e1 and e2 can be in the set.
- 3 concrete classes:
  - `HashSet`, `LinkedHashSet` and `TreeSet`

`HashSet` class

- `HashSet` class is a concrete class that implements `Set`.
- You can create a `HashSet` using its no-arg constructor.
- Its duplicate-free.
- No particular order for the elements in a hash set.
- To impose an order elements, `LinkedHashSet` class can be used.

Cont’d

```
public static void main(String... args){
  Set set = new HashSet();
  String text = "Have a good day. Have a good class. Have a good
   visit. Have fun!";
  StringTokenizer st = new StringTokenizer(text, “ .!?”);
  while(st.hasMoreTokens()){
   set.add(st.nextToken());
  }
  System.out.println(set);

  //obtain an iterator for the hash set
  Iterator iterator = set.iterator();
  while(iterator.hasNext()){
   System.out.println(iterator.next() + " ");
  }//while
 }
```

```
//Output
[day, Have, a, class, fun, good, visit]
day Have a class fun good visit
```

`LinkedHashSet` class

- `LinkedHashSet` class extends `HashSet` with a linked list implementation that supports an ordering of the elements in the set.
- The elements in a `HashSet` are not ordered, but `LinkedHashSet`'s are ordered in which they are inserted.

Cont’d

```
public static void main(String... args){
  Set set = new LinkedHashSet();
  String text = "Have a good day. Have a good class. Have a good
   visit. Have fun!";
  StringTokenizer st = new StringTokenizer(text, “ .!?”);
  while(st.hasMoreTokens()){
   set.add(st.nextToken());
  }
  System.out.println(set);

  //obtain an iterator for the hash set
  Iterator iterator = set.iterator();
  while(iterator.hasNext()){
   System.out.println(iterator.next() + " ");
  }//while
 }
```

```
//Output
[Have, a , good, day, class, visit, fun]
Have a good day class visit fun
```

`TreeSet` class

- `TreeSet` is a concrete class that implement `SortedSet` interface.
- You can use its no-arg constructor or `new TreeSet(Collection)`.
- You can add objects into a tree set as long as they can be compared with each other.
- They are two ways to compare objects:

Cont'd

1. Use the `Comparable` interface.
   - They can be compared using `compareTo` method.
   - For example, `String` class and all the wrapper classes for primitive types, implement the `Comparable` interface.
2. Specify a comparator for the elements in the set.
   - Create a custom Comparator class.

Cont’d

```
public static void main(String... args){
   Set set = new HashSet();
   String text = "Have a good day. Have a good class. Have a good
    visit. Have fun!";
   StringTokenizer st = new StringTokenizer(text, “ .!?”);
   while(st.hasMoreTokens()){
    set.add(st.nextToken());
   }
   TreeSet treeSet = new TreeSet(set);
   System.out.println(treeSet);

   //obtain an iterator for the hash set
   Iterator iterator = treeSet.iterator();
   while(iterator.hasNext()){
    System.out.println(
        iterator.next() + " ");
   }//while
```

```
//Output
[Have, a, class, day, fun, good, visit]
Have a class day fun good visit
```

Cont'd

- `HashSet` performs more efficient to insert and remove elements.
- A runtime error (`ClassCastException`) may be happened if you add an object that is not comparable with the existing objects in the tree set.

Enhanced for-loop

```
Iterator iterator = set.iterator();
  while(iterator.hasNext()){
   System.out.println(iterator.next() + " ");
  }//while
```

You can simplify the code using enhanced for loop without using iterator.

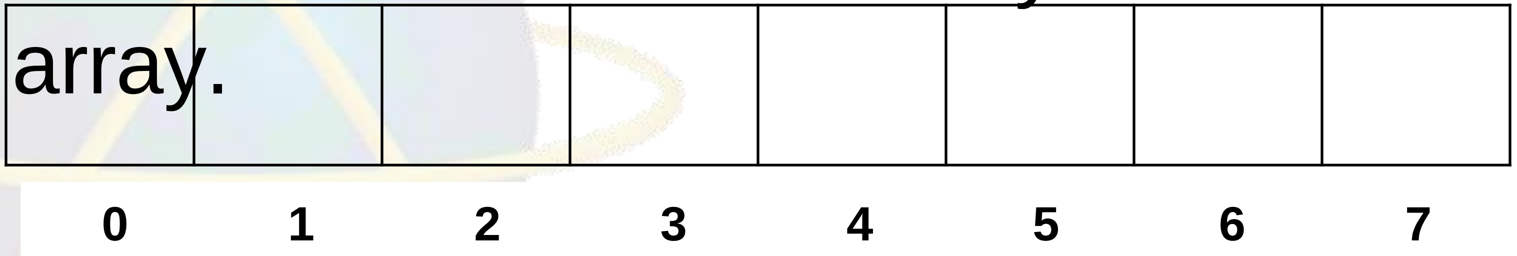
```
for(Object e : set){
 System.out.println(e.toString() + " ");
}
```

`List` interface

- To allow <u>duplicate elements </u>to be stored in a collection.
- To access elements by an index.
- `List` interface extends `Collection` to define an ordered collection with duplicate allowed.

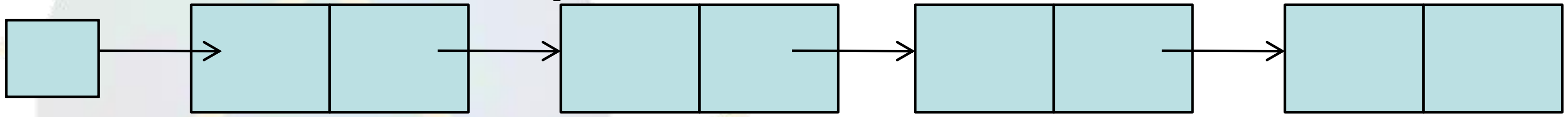
`ArrayList` class

- `ArrayList` stores elements in an array.
- The array is dynamically created.
- If the capacity of the array is exceeded, create a large new array and copy all the elements from current array to the new array.



`LinkedList` class

- `LinkedList` stores elements in a linked list.
- Node reference implementation where Node class has two portion of data (data and nextRef).



Application

- Which of the two classes you use depends on your specific needs.
- If you need to support random access through an index without inserting and removing elements, `ArrayList` is more efficient collection.
- If application requires insertion and deletion of elements in the list, `LinkedList` is suitable.

Vector class

- In Java 2, Vector class is the same as ArrayList, except that it contains synchronized methods for accessing and modifying vector.
- Synchronized method can prevent data corruption when a vector is accessed or modified by two or more threads concurrently.
- Use ArrayList class if you don't need synchronization. It works much faster.

`Stack` class

- Stack is implemented as an extension of Vector.
- Basic operations:
  - isEmpty() – returns boolean if stack is empty.
  - peek() – returns the top element of the stack without removing it.
  - pop() – removes the top element from the stack and returns it.

Cont'd

- push() – adds the specified element to the stack.
- search() – check the specified element is in the stack.
- etc.

Using Generic Type

- Introduced in Java 5.
- Support type checking at compile time.
- Object type is determined when the `Collection` object is constructed.
- E.g.,

```
HashSet<String> set = new HashSet<String>();
```

Cont’d

- if you attempt to add non-string, a compile time error would occur.

```
set.add(new Integer(2));
```

Cont'd

- No casting is required. E.g.,

```
ArrayList<Double> list = new
   ArrayList<Double>();
list.add(5.5);
list.add(3.2);
Double doubleObj = list.get(0);
double d = list.get(1);
```

`Map` interface

- `Map` interface maps keys to the elements.
- The keys are like indexes.
- In `List`, indexes are Integers.
- In `Map`, keys can be any objects.
- A map <u>cannot</u> contain duplicate keys.
- Each key maps to one value.
- Operations: querying, updating, and obtaining a collection of values and a set of keys.

Cont’d

- Update methods include:
  - `clear` – removes all the mappings
  - `put` – associates the specified value with the specified key
  - `putAll` – adds the specified map to this map
  - `remove` – remove the map elements for the specified key

Cont'd

- Query methods include:
  - `containsKey` – checks whether the map contains a mapping for the specified key
  - `containsValue` – checks whether the map contains a mapping for this value
  - `isEmpty` – checks the maps contains any mappings.
  - `size` – returns the number of mappings in the map

Cont’d

- More methods:
  - `keySet`
  - `values`
  - `entrySet`
  - Etc.

Cont’d

- Classes:
  - `HashMap`,
  - `LinkedHashMap`,
  - `TreeMap`
- `HashMap` class is efficient for locating a value, inserting a mapping and deleting a mapping.
- `LinkedHashMap` (Java 4) extends `HashMap` with linked list implementation that supports an ordering of the entries in

Cont'd

- `TreeMap` implements `SortedMap`, is efficient for traversing the keys in a sorted order.
- The keys are sorted using the `Comparable` or `Comparator` interface.

Cont’d

- E.g.,

```
HashMap hashMap = new HashMap();
hashMap.put(“Smith”, new Loan(7, 15, 120000));
hashMap.put(“Anderson”, new Loan(9, 30, 200000));
hashMap.put(“Lewis”, new Loan(2, 25, 125000));
```

`Collections` class

- `Collections` class contains <u>static methods</u> for operating on collections and maps.
- Most of the methods deal with lists.
- `sort` methods can be used to sort a list using the `Comparable` interface or the `Comparator` interface.
- More methods…

`Arrays` class

- `Arrays` class contains various static methods for sorting and searching arrays, comparing arrays, and filling arrays elements.
- It also contains a method for converting an array to a list.

Quick Review Questions

- Describe Java Collection Framework.
- How do you create an instance of Set?
- What are the differences between HastSet, LinkedHastSet and TreeSet?
- How do you traverse the elements in a set?
- List the implemented classes of List interface.
- Describe and compare HashMap, LinkedHashMap, and TreeMap

Summary of Main Teaching Points

- Collections
-Hash set
-Tree set
-List
-ArrayList
-Vector

Q & A

Next Session

- UML diagrams
  - -Usecase diagram
  - -Class diagram
  - -Activity diagram