JSpiders
Training & Development Center

Java Collection Framework

Dr. Shishira Bhat

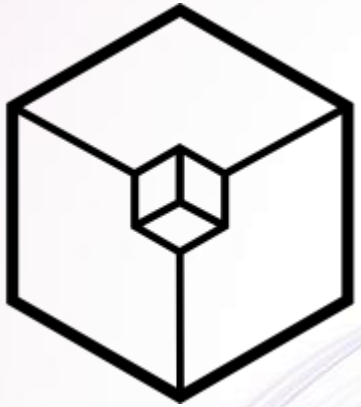
Topics

Core Java

JSpiders
Training & Development Center

Collection Framework

Framework



- It is an abstraction which defines generic or common functionalities or utilities.
- This generic functionalities can be extended and used or can be modified by additional user-written code.
- Mainly used for application development.
- They are tested.
- Uses best practices and necessary design patterns.
- Different frameworks are available for different kinds of application development.

Collection Framework

HIBERNATE LOG4J™ JUnit 5

JSpiders
Training & Development Center

Collection Framework

What is a Collection Framework?

- Framework that deals with collection/group of data.
- It has predefined interfaces, implementation classes, utility methods which are collectively present in ***`java.util`*** package.

Why Collection Framework?

- For scalability.
- To store heterogeneous data.
- Effective way for data handling.
- Defined data structures and ready made methods.

JSpiders
Training & Development Center

Arrays Vs Collection

JSpiders
Training & Development Center

Arrays	Collection
Homogenous in nature wherein only one type of data can be stored.	Heterogeneous in nature wherein multiple types of data can be stored.
It is static or fixed in size.	It is dynamic or elastic.
Doesn’t have any inbuilt methods for data handling.	Has many inbuilt methods for data handling.
Can store primitive data.	Can’t store primitive data.
Doesn’t use any data structure.	Uses data structures.
In case of non-primitive data, an array can store only reference.	It can store object as well as reference.

Topics

Collection Framework

Collection Framework

01 Collection

JSpiders
Training & Development Center

What's possible?

JSpiders
Training & Development Center

Collection can't store primitive data.

In order to add any of primitive data into collection, then it has to be done through its corresponding Wrapper class object.

20	20.5	A	JSpiders	true

Example:

Storing primitive data as Wrapper Class Object (autoboxing).

```
Collection col = ArrayList();
Col.add(20);
```

Example:

Storing primitive data as Wrapper Class Object.

```
Collection col = ArrayList();
Integer inp = new Integer(20);
Col.add(inp);
```

Methods of Collection Interface

```
int size();
boolean isEmpty();
boolean contains(Object o);
Object[] toArray();
boolean add(E e);
boolean remove(Object o);
boolean containsAll(Collection<?> c);
boolean addAll(Collection<? extends E> c);
boolean removeAll(Collection<?> c);
boolean retainAll(Collection<?> c);
void clear();
```

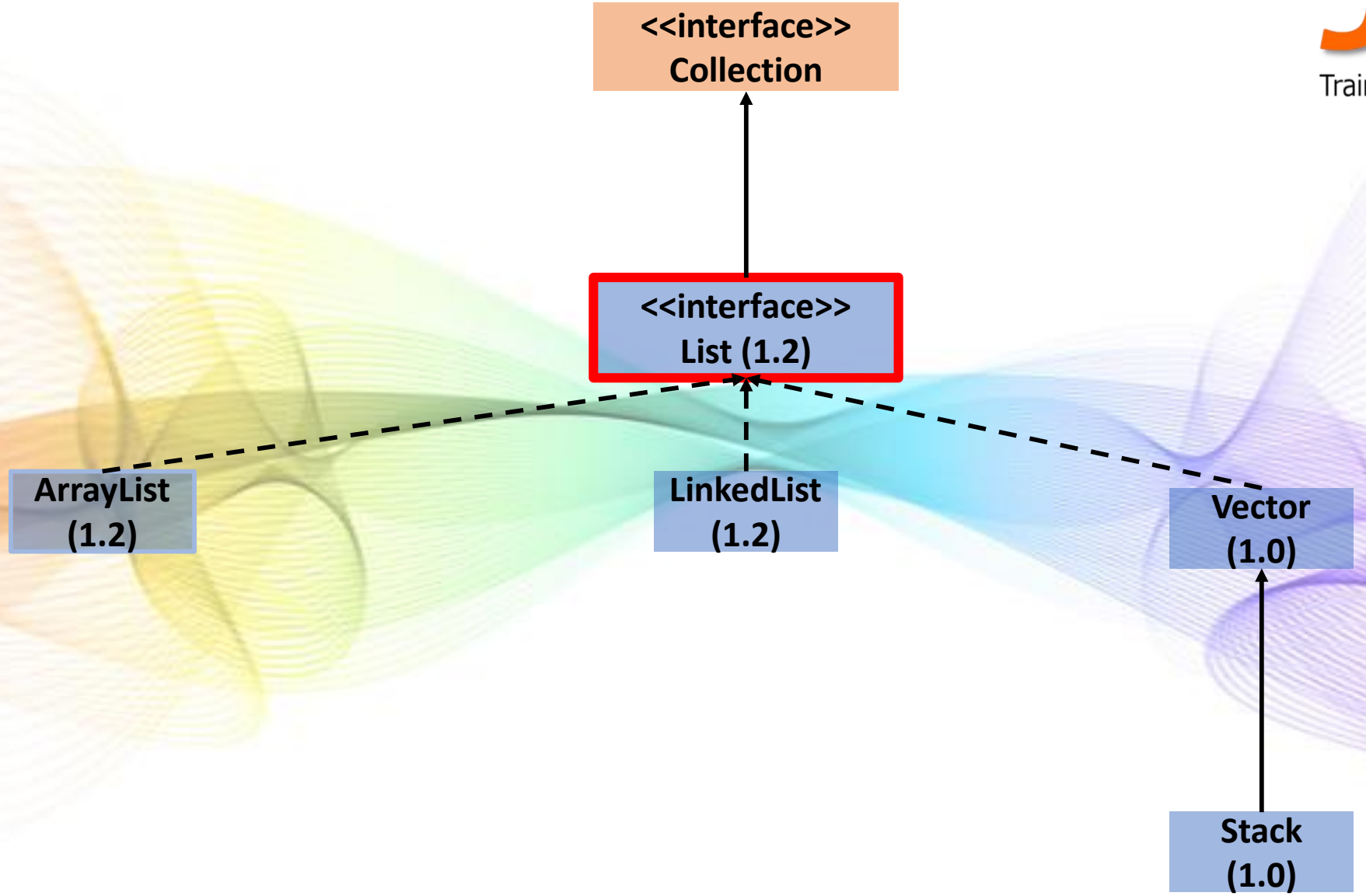
Topics

Collection Framework

Collection Framework

01 List

JSpiders
Training & Development Center



List Interface

- List Interface is the sub interface of Collection.
- It contains index-based methods to insert and delete elements.

List Interface declaration

- public interface List<E> extends Collection<E>

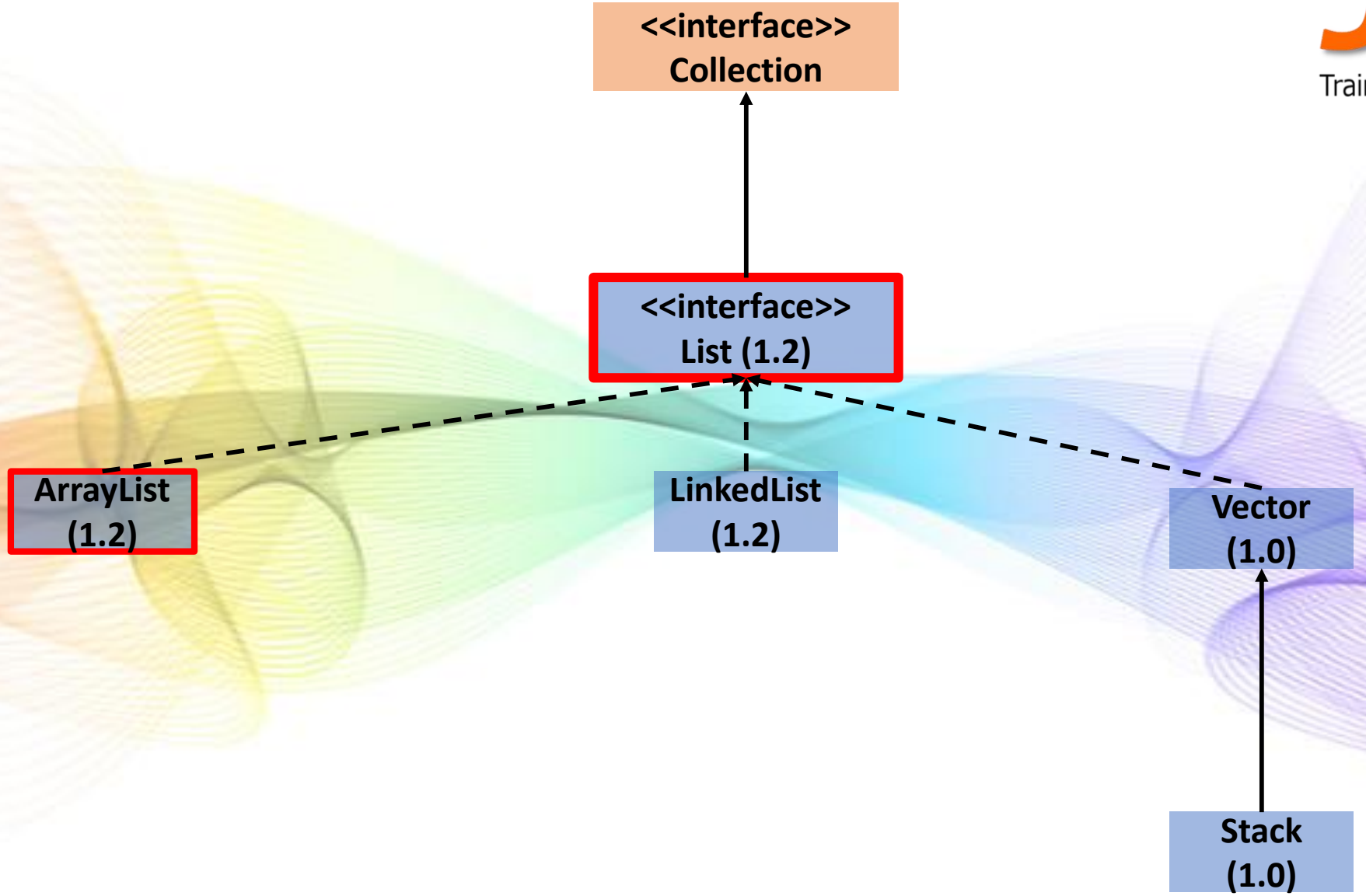
Topics

Collection Framework

Collection Framework

02 ArrayList

JSpiders
Training & Development Center



Array List

- ArrayList is not a legacy class. It is introduced in JDK 1.2.
- ArrayList can contain duplicate elements.
- ArrayList maintains insertion order.
- ArrayList is non synchronized.
- ArrayList allows random access because array works at the index basis.
- ArrayList internally uses a dynamic array (growable/resizable) to store the elements.
- Data is stored in the form of array.

Array List

- Initial Capacity and Incremental Capacity is applicable.
- If the total number of elements exceeds than its capacity, the incremental capacity of the ArrayList is ((Current Capacity * 3)/2) + 1
- Cannot control the growth. That is the incremental capacity is fixed.
- Involves shift operations. That is manipulation with ArrayList is slow because it internally uses an array.
- If any element is removed from the array, all the bits are shifted in memory.
- An ArrayList class can act as a list only because it implements List only.

Array List

- ArrayList is better for storing and accessing data.
- Consumes less memory.
- Memory is continuous.
- Has three overloaded constructors:
- public ArrayList(int initialCapacity)
- public ArrayList()
- public ArrayList(Collection<? extends E> c)

Array List

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.

Array List

Method	Description
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
boolean contains(Object o)	It returns true if the list contains the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
protected void removeRange(int fromIndex, int toIndex)	It is used to remove all the elements lies within the given range.
void replaceAll(UnaryOperator<E> operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.

Array List

Method	Description
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
int size()	It is used to return the number of elements present in the list.

Array vs Array List

Basis	Array	ArrayList
Resizable	Fixed in size	Resizable
Performance	Fast	Slow
Primitives	Possible to store primitive data types	Not possible to store primitive data types
Iterating the values	Can use for loop, for each loop	Can use iterator, for loop and for each loop
Length	length variable provides the length of the array.	The size method provides the length of the ArrayList.
Adding elements	The assignment operator is used in an array in order to add elements.	The add method is used in an ArrayList object in order to add elements.
Multi-dimensional	An array can be either a single dimensional array or a multidimensional array.	But an ArrayList is only single dimensional.

Array List

Scenario

Write a java program that contains an Employee class with the following properties. eid (type is int), ename (type is String), Department (type is Department).

Now create another class named Department which contains the properties did(type is int), dname(type is String) and designation (type is String).

Now create another class named EmployeeApp which contains the main method.

Now write code that prompts the user to enter the Employee details like id, name, and prompt Department details like department id, name and designation details in ArrayList<Employee> object.

Enter at least 3 employees and department info print those details to the output.

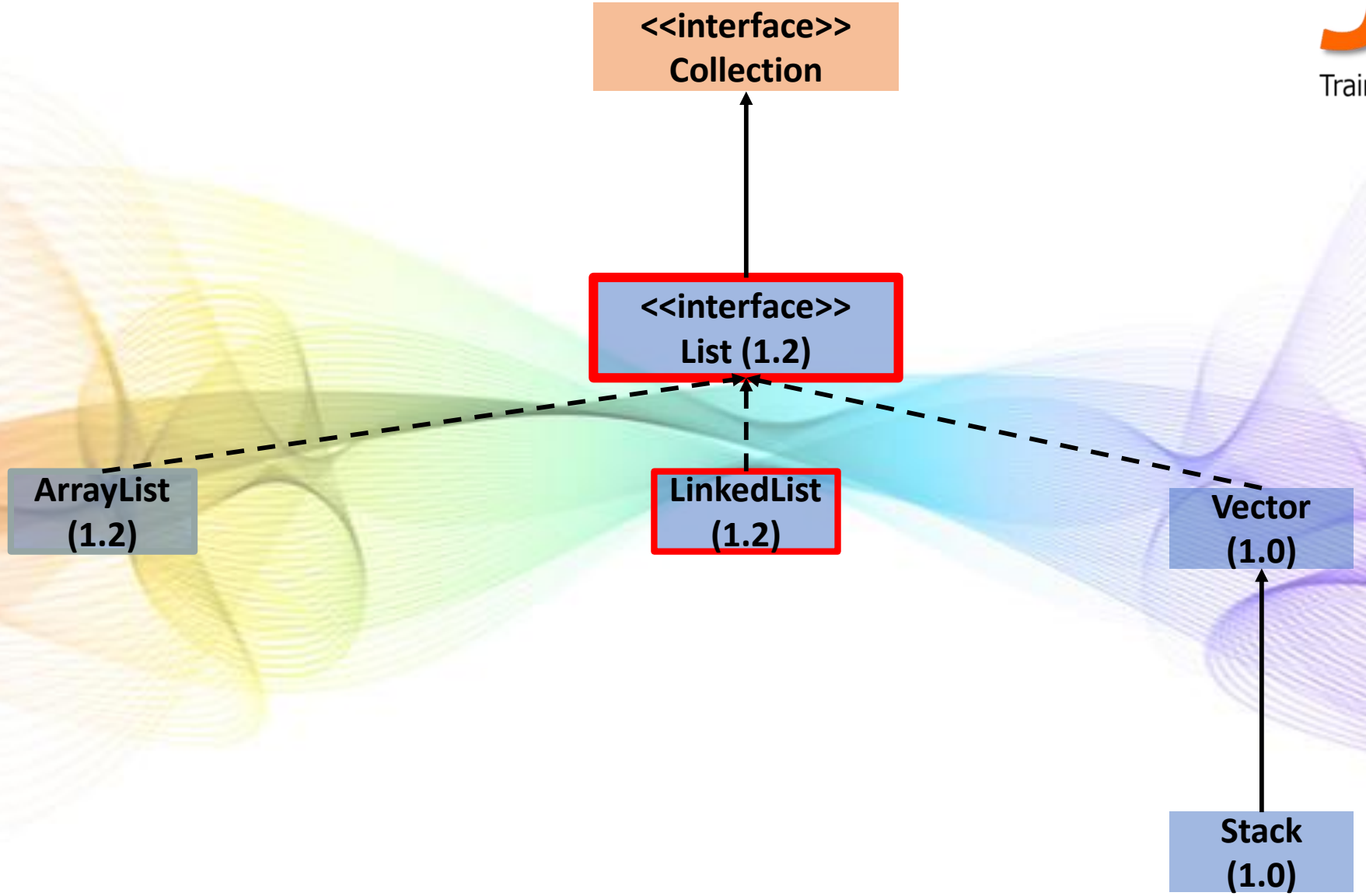
Topics

Collection Framework

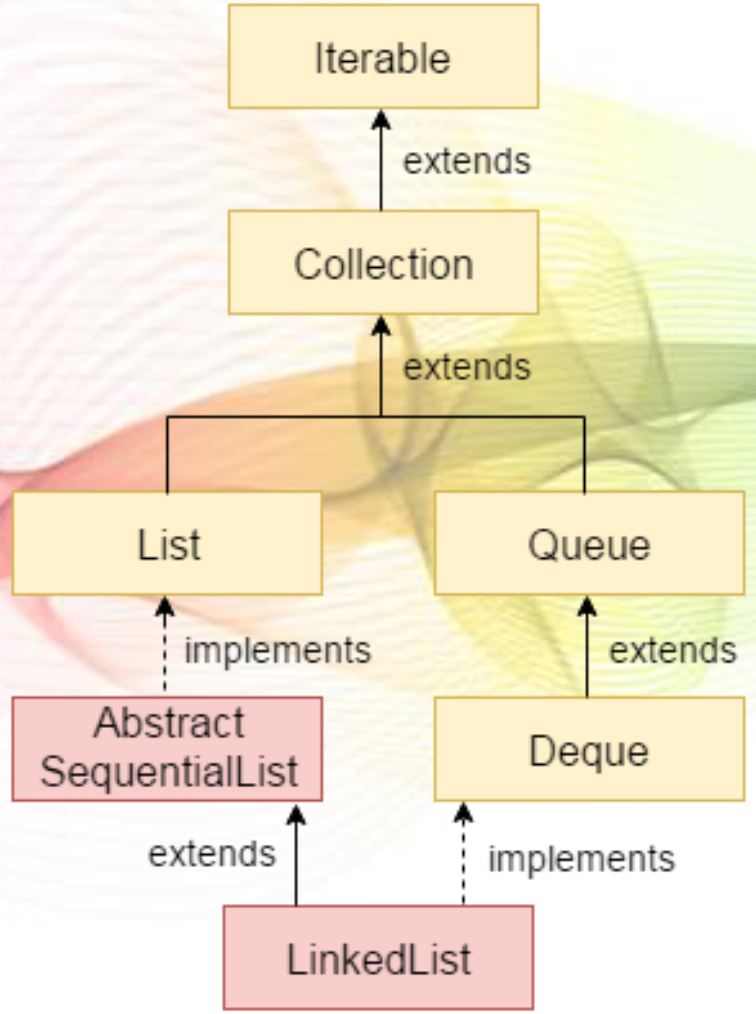
Collection Framework

03 LinkedList

JSpiders
Training & Development Center



Linked List



- **LinkedList class uses a doubly linked list to store the elements.**
- **It provides a linked-list data structure.**
- **It inherits the AbstractList class and implements List and Deque (Deque - Double Ended Queue) interfaces.**

Linked List

- LinkedList can contain duplicate elements.
- LinkedList maintains insertion order.
- LinkedList is non synchronized.
- In LinkedList, manipulation is fast because no shifting needs to occur.
- LinkedList can be used as a list, stack or queue.
- Has two Constructors:
  - LinkedList()
  - LinkedList(Collection<? extends E> c)

Linked List

Method	Description
boolean add(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
void clear()	It is used to remove all the elements from a list.
boolean contains(Object o)	It is used to return true if a list contains a specified element.
E get(int index)	It is used to return the element at the specified position in a list.

Linked List

Method	Description
E getFirst()	It is used to return the first element in a list.
E getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.
boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.

Linked List

Method	Description
E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
E pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.

Linked List

Method	Description
boolean removeFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
int size()	It is used to return the number of elements in a list.

Array List vs Linked List

ArrayList	LinkedList
ArrayList internally uses a dynamic array (growable/resizable) to store the elements.	LinkedList internally uses a doubly linked list to store the elements.
Data is stored in the form of array.	Data is stored in the form of node.
Initial Capacity and Incremental Capacity is applicable.	Initial Capacity and Incremental Capacity is not applicable.
Involves shift operations. That is manipulation with ArrayList is slow because it internally uses an array and If any element is removed from the array, all the bits are shifted in memory.	Does not involves any shift operations. That is manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
Has three overloaded constructors.	Has two overloaded constructors.
Memory is continuous.	Memory may not be continuous.
An ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data. (Insertion or removal of data in between)
Consumes less memory.	Consumes more memory.

Linked List

Scenario

Write a java program to implement Linked List to Make A Playlist.

Write a java program to implement Linked List to navigate to Previous and next page in web browser.

Write a java program to implement Linked List to navigate within the Lift of a given building

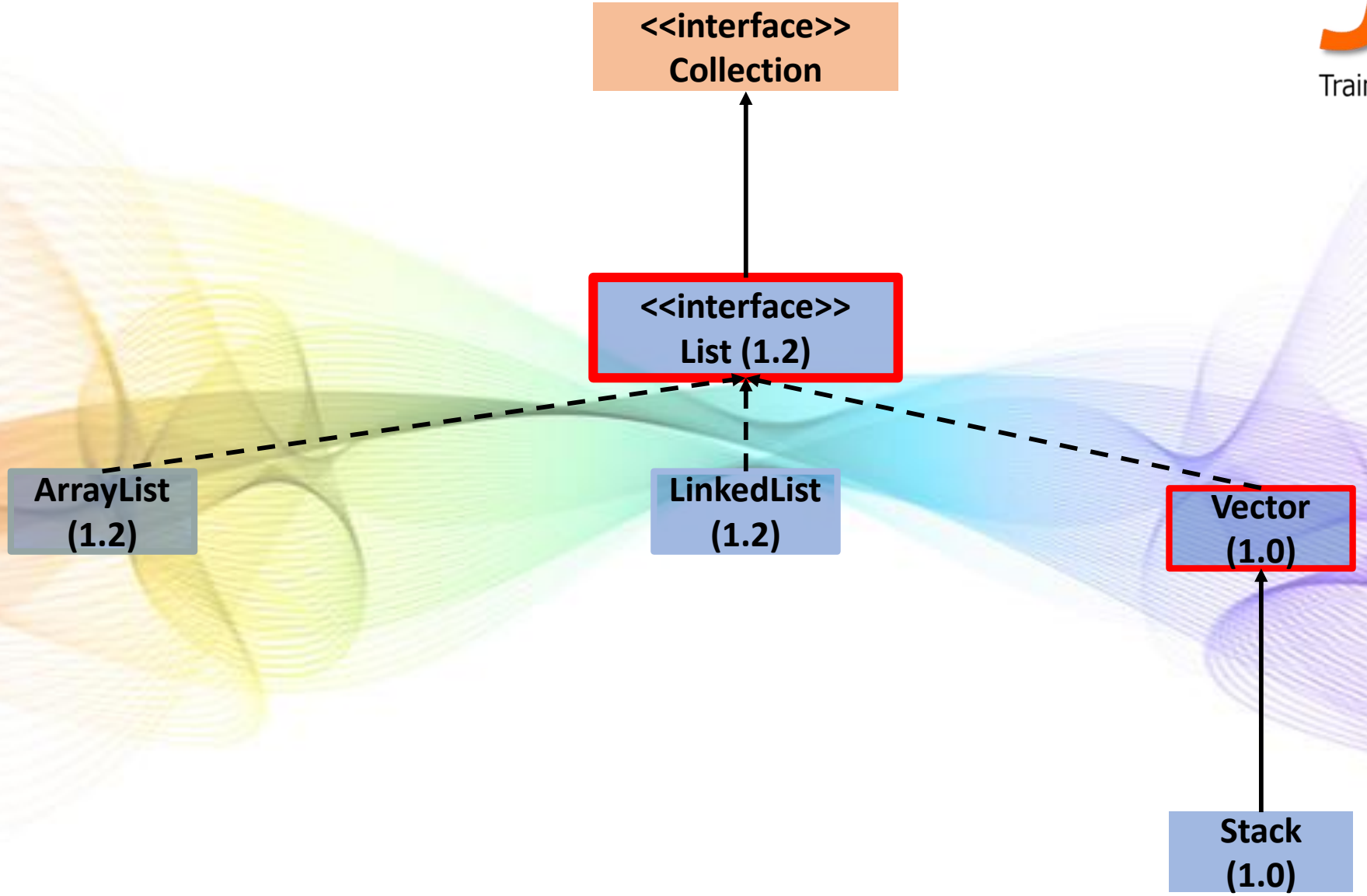
Topics

Collection Framework

Collection Framework

04 Vector

JSpiders
Training & Development Center


<<interface>>
Collection
<<interface>>
List (1.2)
ArrayList
(1.2)
LinkedList
(1.2)
Vector
(1.0)
Stack
(1.0)

Vector

- Vector class comes under the java.util package.
- Vector implements a growable array of objects.
- Like an array, it contains the component that can be accessed using an integer index.
- Vector is very useful if we don't know the size of an array in advance or we need one that can change the size over the lifetime of a program.
- Vector implements a dynamic array that means it can grow or shrink as required.
- It is similar to the ArrayList, but with two differences-
    - Vector is synchronized.
    - The vector contains many legacy methods that are not the part of a collections framework

Vector

- Vector class supports four types of constructors.

- vector()
  - – default size will be 10 if this is used.
- vector(int initialCapacity)
  - – initial size is defined by the user, incremental size is 0.
- vector(int initialCapacity, int capacityIncrement)
  - – both initial size & incremental size is defined by the user.
- Vector(Collection<? extends E> c)
  - – creates vector that contains elements of a collection.

Vector

Method	Description
add()	It is used to append the specified element in the given vector.
addAll()	It is used to append all of the elements in the specified collection to the end of this Vector.
addElement()	It is used to append the specified component to the end of this vector. It increases the vector size by one.
capacity()	It is used to get the current capacity of this vector.
clear()	It is used to delete all of the elements from this vector.
clone()	It returns a clone of this vector.
contains()	It returns true if the vector contains the specified element.
containsAll()	It returns true if the vector contains all of the elements in the specified collection.
copyInto()	It is used to copy the components of the vector into the specified array.
elementAt()	It is used to get the component at the specified index.
elements()	It returns an enumeration of the components of a vector.

Vector

Method	Description
ensureCapacity()	It is used to increase the capacity of the vector which is in use, if necessary. It ensures that the vector can hold at least the number of components specified by the minimum capacity argument.
equals()	It is used to compare the specified object with the vector for equality.
firstElement()	It is used to get the first component of the vector.
forEach()	It is used to perform the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
get()	It is used to get an element at the specified position in the vector.
hashCode()	It is used to get the hash code value of a vector.
indexOf()	It is used to get the index of the first occurrence of the specified element in the vector. It returns -1 if the vector does not contain the element.
insertElementAt()	It is used to insert the specified object as a component in the given vector at the specified index.
isEmpty()	It is used to check if this vector has no components.
iterator()	It is used to get an iterator over the elements in the list in proper sequence.
lastElement()	It is used to get the last component of the vector.

Vector

Method	Description
lastIndexOf()	It is used to get the index of the last occurrence of the specified element in the vector. It returns -1 if the vector does not contain the element.
listIterator()	It is used to get a list iterator over the elements in the list in proper sequence.
remove()	It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged.
removeAll()	It is used to delete all the elements from the vector that are present in the specified collection.
removeAllElements()	It is used to remove all elements from the vector and set the size of the vector to zero.
removeElement()	It is used to remove the first (lowest-indexed) occurrence of the argument from the vector.
removeElementAt()	It is used to delete the component at the specified index.
removeIf()	It is used to remove all of the elements of the collection that satisfy the given predicate.
removeRange()	It is used to delete all of the elements from the vector whose index is between fromIndex, inclusive and toIndex, exclusive.
replaceAll()	It is used to replace each element of the list with the result of applying the operator to that element.
retainAll()	It is used to retain only that element in the vector which is contained in the specified collection.
set()	It is used to replace the element at the specified position in the vector with the specified element.
setElementAt()	It is used to set the component at the specified index of the vector to the specified object.

Vector

Method	Description
setSize()	It is used to set the size of the given vector.
size()	It is used to get the number of components in the given vector.
sort()	It is used to sort the list according to the order induced by the specified Comparator.
spliterator()	It is used to create a late-binding and fail-fast Spliterator over the elements in the list.
subList()	It is used to get a view of the portion of the list between fromIndex, inclusive, and toIndex, exclusive.
toArray()	It is used to get an array containing all of the elements in this vector in correct order.
toString()	It is used to get a string representation of the vector.
trimToSize()	It is used to trim the capacity of the vector to the vector's current size.

ArrayList vs Vector

ArrayList	Vector
ArrayList is not synchronized. That is ArrayList is multi-threaded.	Vector is synchronized. That is Vector is single threaded.
If the total number of elements exceeds than its capacity, the incremental capacity of the ArrayList is ((Current Capacity * 3)/2) + 1	Vector increments 100% which means it doubles the array size if the total number of elements exceeds than its capacity. That is **Current Capacity * 2**
ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a legacy class. It is present since JDK 1.0.
ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
ArrayList uses the Iterator interface to traverse the elements.	A Vector can use the Iterator interface or Enumeration interface to traverse the elements.
Has three overloaded constructors.	Has four overloaded constructors.
Cannot control the growth. That is the incremental capacity is fixed.	Can control the growth. That is the incremental capacity can be explicitly mentioned by the programmer. It is done using one of the overloaded constructors. **public vector (int initialCapacity, int incrementalCapacity)**

Topics

Collection Framework

Collection Framework

05 Stack

JSpiders
Training & Development Center

Stack

- Java Collection framework provides a Stack class which models and implements Stack data structure.
- The class is based on the basic principle of last-in-first-out.
- In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek.

Stack

Method	Description
Object push(Object element)	Pushes an element on the top of the stack.
Object pop()	Removes and returns the top element of the stack. An 'EmptyStackException' exception is thrown if we call pop() when the invoking stack is empty.
Object peek()	Returns the element on the top of the stack, but does not remove it.
boolean empty()	It returns true if nothing is on the top of the stack. Else, returns false.
int search(Object element)	It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.

Topics

Collection Framework

Collection Framework

06 Set

JSpiders
Training & Development Center

Set

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

Set

- **Union - addAll**
- **Intersection - retainAll**
- **Difference - removeAll**

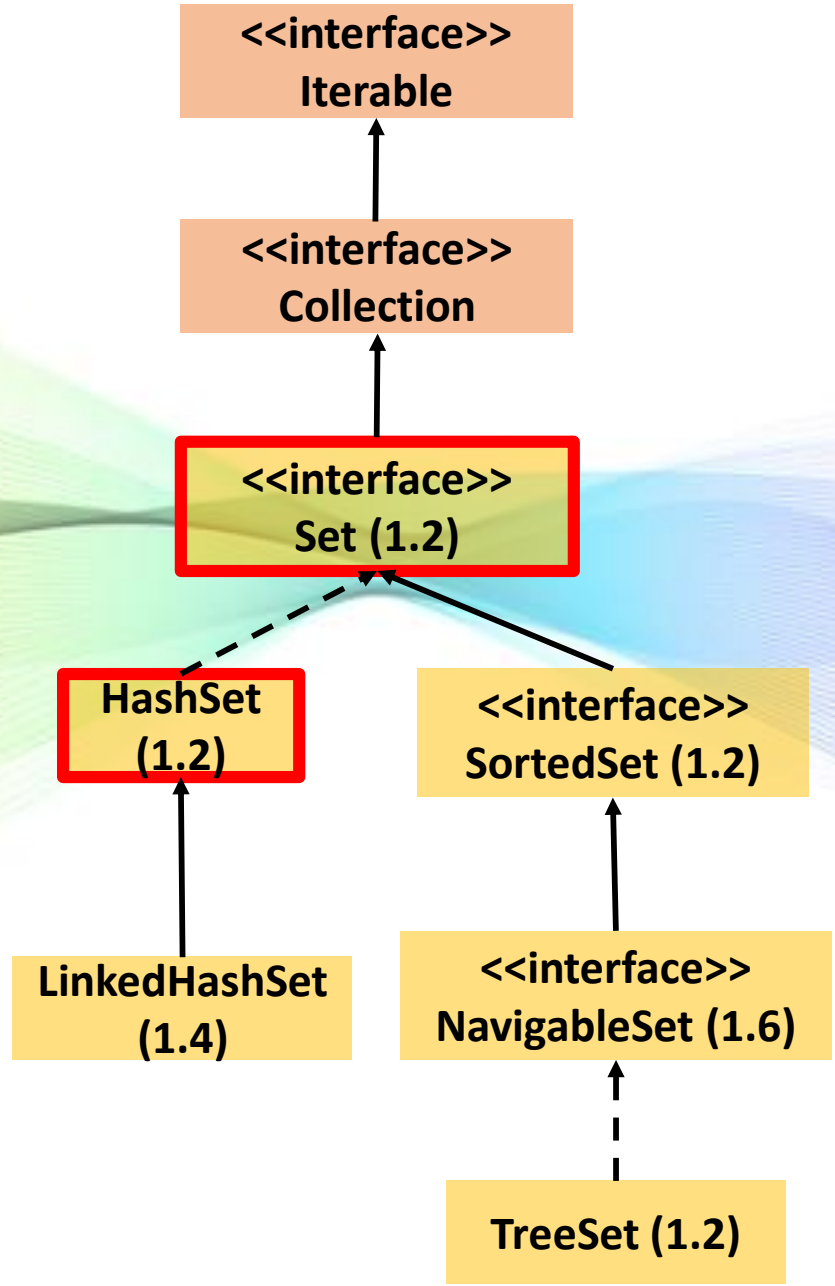
Topics

Collection Framework

Collection Framework

07 HashSet

JSpiders
Training & Development Center



Hash Set

- Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
- HashSet stores the elements by using a mechanism called hashing.
- HashSet contains unique elements only.
- HashSet allows one null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.

Hash Set

- The initial default capacity of HashSet is 16, and the load factor is 0.75.
- It has 4 constructors
  - HashSet() – It is used to construct a default HashSet.
  - HashSet(int capacity) – It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
  - HashSet(int capacity, float loadFactor) – It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
  - HashSet(Collection<? extends E> c) – It is used to initialize the hash set by using the elements of the collection c.

Hash Set

Method	Description
Boolean add(E e)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from the set.
object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean isEmpty()	It is used to return true if this set contains no elements.
Iterator<E> iterator()	It is used to return an iterator over the elements in this set.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present
Int size()	It is used to return the number of elements in the set.

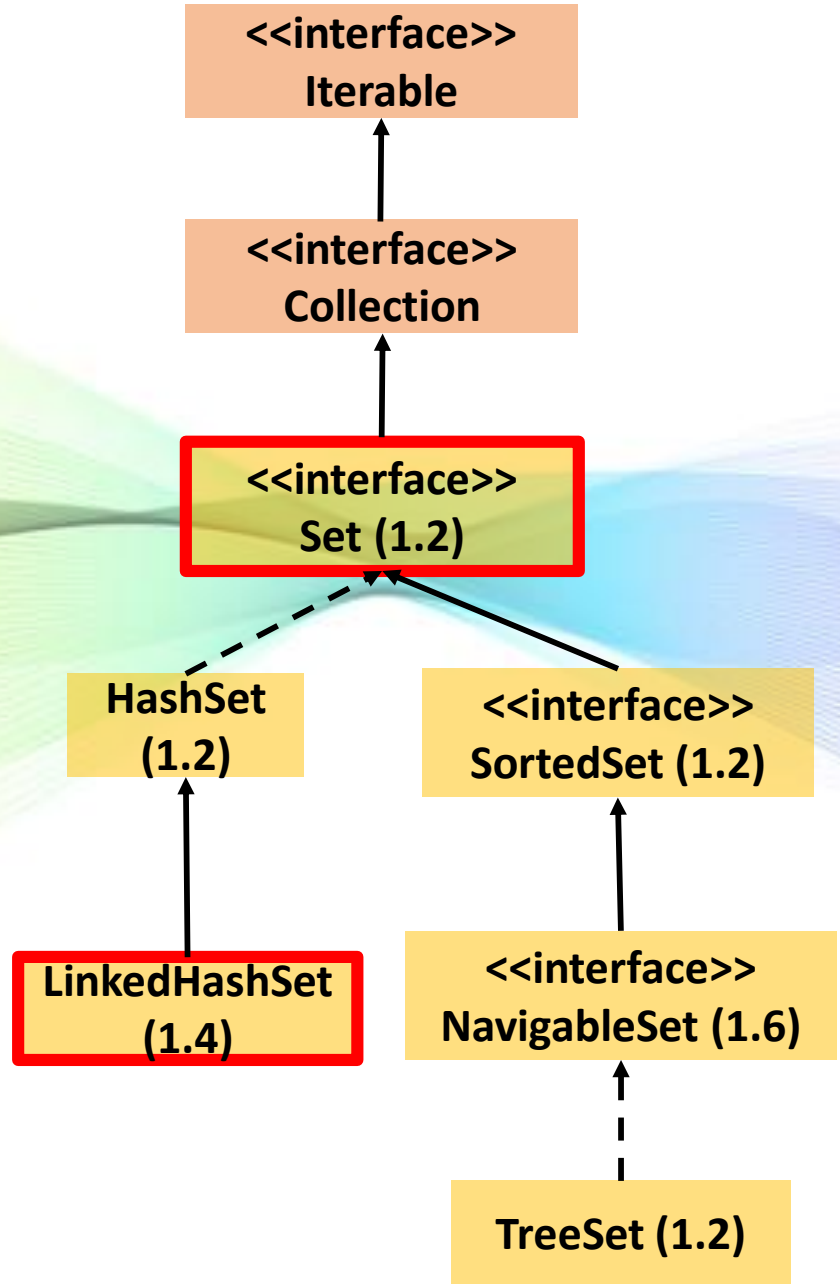
Topics

Collection Framework

Collection Framework

08 LinkedHashSet

JSpiders
Training & Development Center



LinkedHash Set

- Java LinkedHashSet class is used to create a collection that uses a hash table for storage. It extends HashSet and implements Set interface.
- LinkedHashSet stores the elements by using a mechanism called hashing.
- LinkedHashSet contains unique elements only.
- LinkedHashSet allows null value.
- HashSet class is non synchronized.
- LinkedHashSet maintain the insertion order.
- HashSet is the best approach for insertion and removal operations.

LinkedHash Set

- The initial default capacity of HashSet is 16, and the load factor is 0.75.
- It has 4 constructors
  - LinkedHashSet() – It is used to construct a default HashSet.
  - LinkedHashSet(int capacity) – It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
  - LinkedHashSet(int capacity, float loadFactor) – It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
  - HashSet(Collection<? extends E> c) – It is used to initialize the hash set by using the elements of the collection c.

LinkedHash Set

Method	Description
Boolean add(E e)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from the set.
object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean isEmpty()	It is used to return true if this set contains no elements.
Iterator<E> iterator()	It is used to return an iterator over the elements in this set.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present
Int size()	It is used to return the number of elements in the set.

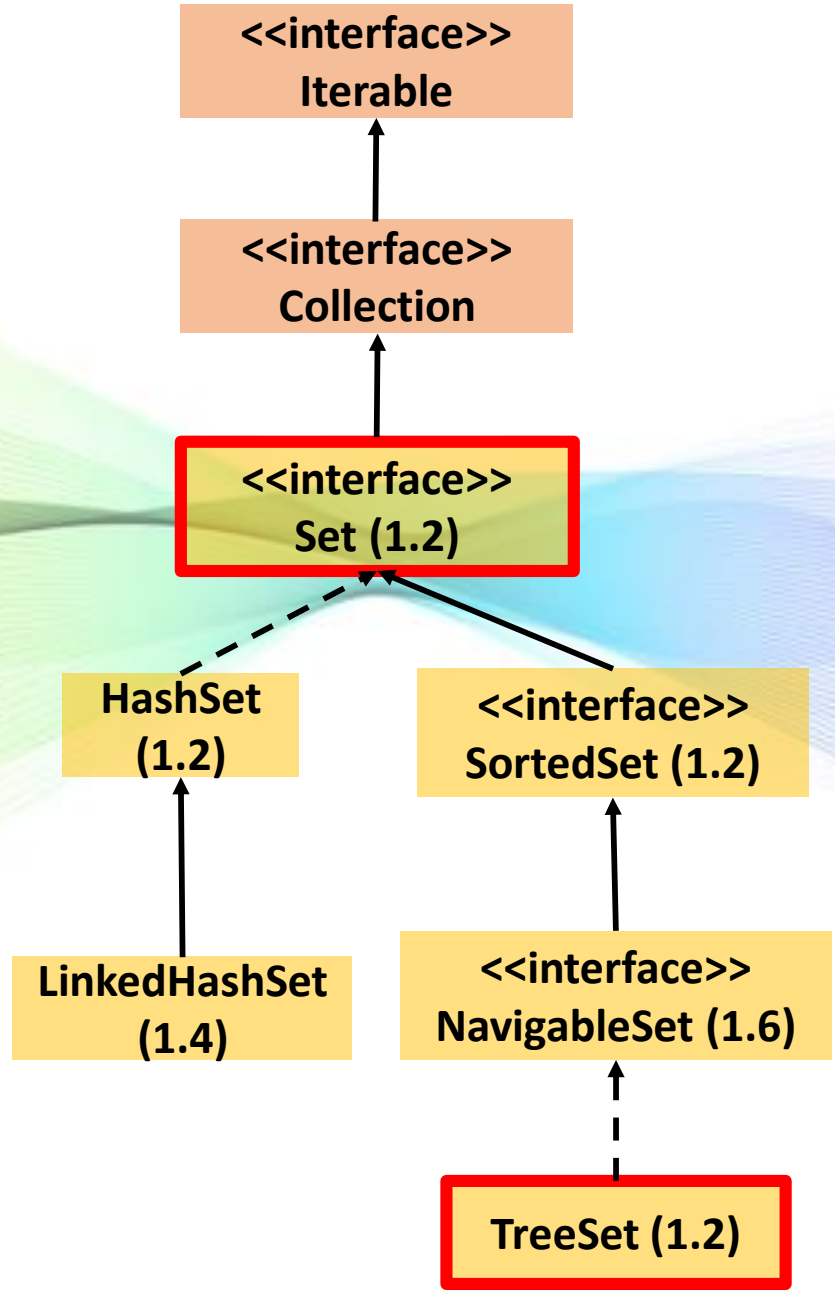
Topics

Collection Framework

Collection Framework

09 TreeSet

JSpiders
Training & Development Center



Tree Set

- **TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage.**
- **The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided.**
- **The TreeSet implements a NavigableSet interface by inheriting AbstractSet class.**

Tree Set

- TreeSet implements the SortedSet interface so duplicate values are not allowed.
- Objects in a TreeSet are stored in a sorted and ascending order.
- TreeSet does not preserve the insertion order of elements.
- TreeSet serves as an excellent choice for storing large amounts of sorted information which are supposed to be accessed quickly because of its faster access and retrieval time.
- TreeSet is basically implementation of a self-balancing binary search tree.

Tree Set

Constructors of TreeSet class:

- **TreeSet t = new TreeSet();**
  This will create empty TreeSet object in which elements will get stored in default natural sorting order.

- **TreeSet t = new TreeSet(Comparator comp);**
  This constructor is used when external specification of sorting order of elements is needed.

- **TreeSet t = new TreeSet(Collection col);**
  This constructor is used when any conversion is needed from any Collection object to TreeSet object.

- **TreeSet t = new TreeSet(SortedSet s);**
  This constructor is used to convert SortedSet object to TreeSet Object.

Tree Set

Method	Description
void add(Object o):	This method will add specified element according to some sorting order in TreeSet. Duplicate entires will not get added.
boolean addAll(Collection c):	This method will add all elements of specified Collection to the set. Elements in Collection should be homogeneous otherwise ClassCastException will be thrown. Duplicate Entries of Collection will not be added to TreeSet.
void clear():	This method will remove all the elements.
boolean contains(Object o):	This method will return true if given element is present in TreeSet else it will return false.
Object first():	This method will return first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
Object last():	This method will return last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
SortedSet headSet(Object toElement):	This method will return elements of TreeSet which are less than the specified element.
SortedSet tailSet(Object fromElement):	This method will return elements of TreeSet which are greater than or equal to the specified element.
SortedSet subSet(Object fromElement, Object toElement):	This method will return elements ranging from fromElement to toElement. fromElement is inclusive and toElement is exclusive.

Tree Set

Method	Description
boolean isEmpty():	This method is used to return true if this set contains no elements or is empty and false for the opposite case.
Object clone():	The method is used to return a shallow copy of the set, which is just a simple copied set.
int size():	This method is used to return the size of the set or the number of elements present in the set.
boolean remove(Object o):	This method is used to return a specific element from the set.
Iterator iterator():	Returns an iterator for iterating over the elements of the set.
Comparator comparator():	This method will return Comparator used to sort elements in TreeSet or it will return null if default natural sorting order is used.
ceiling(E e):	This method returns the least element in this set greater than or equal to the given element, or null if there is no such element.
descendingIterator():	This method returns an iterator over the elements in this set in descending order.
descendingSet():	This method returns a reverse order view of the elements contained in this set.

Tree Set

Method	Description
floor(E e):	This method returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
higher(E e):	This method returns the least element in this set strictly greater than the given element, or null if there is no such element.
lower(E e):	This method returns the greatest element in this set strictly less than the given element, or null if there is no such element.
pollFirst():	This method retrieves and removes the first (lowest) element, or returns null if this set is empty.
pollLast():	This method retrieves and removes the last (highest) element, or returns null if this set is empty.
spliterator():	This method creates a late-binding and fail-fast Spliterator over the elements in this set.

HashSet vs LinkedHashSet vs Tree Set

Criteria	HashSet	LinkedHashSet	TreeSet
Duplicates	Not Allowed	Not Allowed	Not Allowed
Non-Synchronized	Yes	Yes	Yes
Speed	First	Second	Third
Ordering	No Order	Insertion Order	Natural Sort Order
Null Values	One Null Allowed	One Null Allowed	Not Allowed
Comparison	uses equals()	uses equals()	uses compareTo()