Slide Note

Store and Manipulate Objects Using Ordered Collections

Menu

00:06:00:00:50

06/ 50

Slide Note

Learning Objectives

- Understand the Collection Framework
- Describe the List Interface
- Implementing ArrayList
- Introduction to Generics
- Implementing Generics in Collections
- Implement LinkedList
- Explore Iterators in Collection
- Storing user-defined Objects in a Collection

Slide Note

Understanding Collection Framework

Slide Note

Collection Framework

- The collection framework provides built-in interfaces and classes to store and manipulate a group of objects.
- A collection in Java:
  - Consists of a group of objects that are built-in or user-defined.
  - Only holds objects and not primitive datatypes.
- Java Collection Framework helps us to do searching, sorting, insertion, manipulation, and deletion on different types of data.
- Java collections can grow dynamically.
- Java collections are in the package `java.util`.

Slide Note

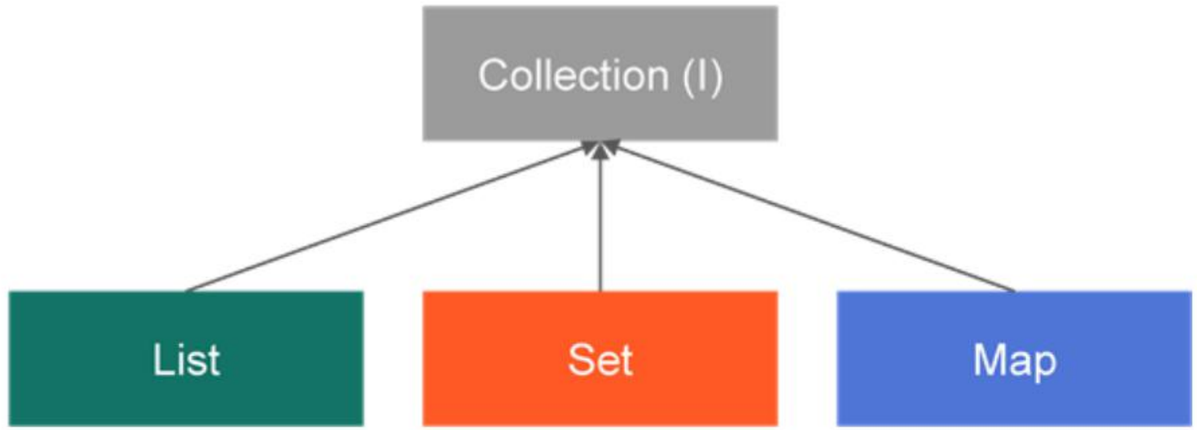
The Need for a Collection Framework

- Arrays store only homogeneous data types and have a fixed size.
- Collections permit heterogeneous storage of data types and are growable in nature.
- The collection framework reduces programming effort by providing useful data structures and algorithms.
- They also increase program speed and quality since the implementation of each interface is interchangeable.

Slide Note

Types of Java Collections

- Here, the collection is an Interface.



- List, Set, Map, Queue, etc. are all interfaces that extend Collection interfaces.
- These are the types of collections that Java provides to store and manipulate objects or data.
- All of the above interfaces have an implementation class that has built-in APIs that a programmer uses.

Note – In this Sprint you will learn about the List interface and a few of its implementation classes.

Slide Note

Here, the collection is a Super interface that extends the Iterable interface.

List, set, dequeue, queue, etc. are the interfaces that extend the Collection interface.

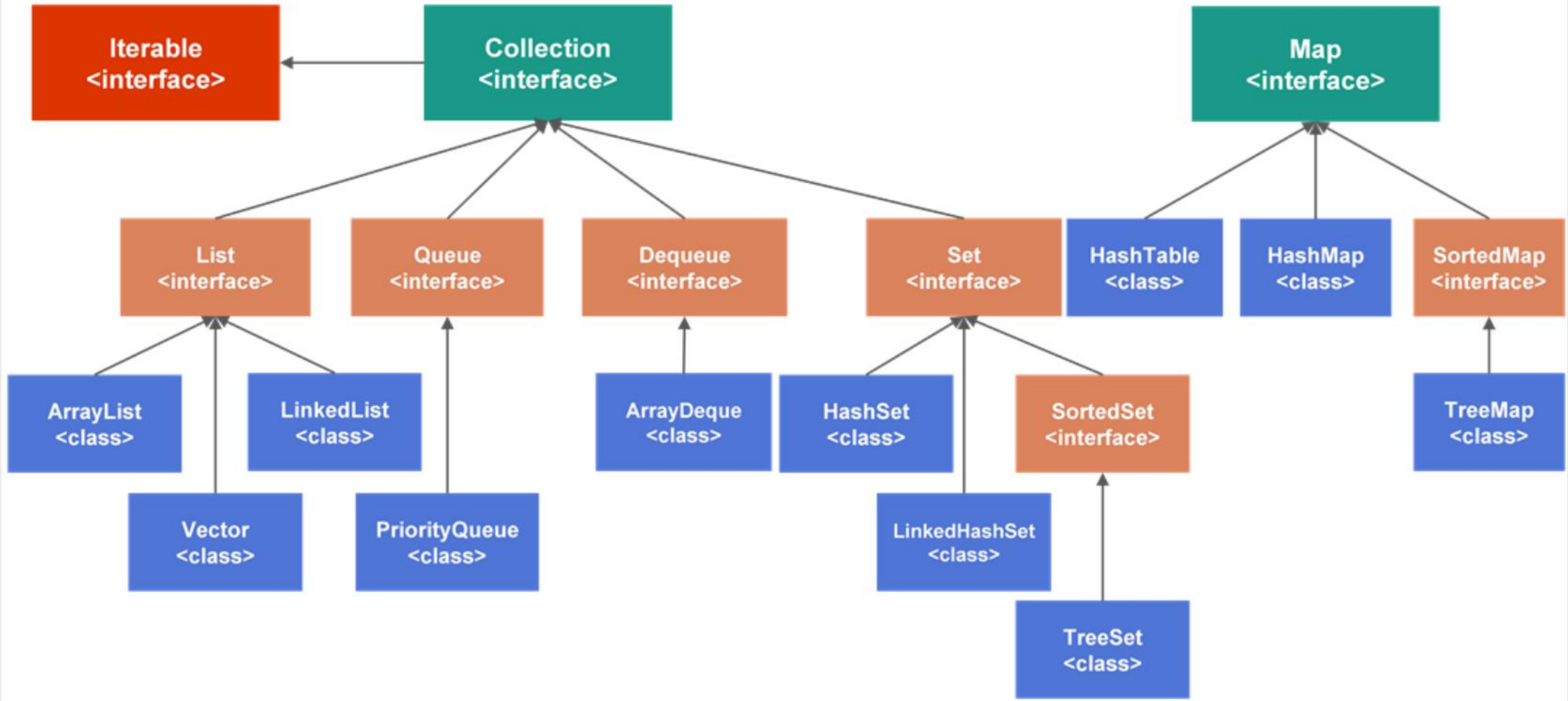
All the interfaces have their own implementation classes.

So for the list, the implementation classes are ArrayList, LinkedList, and Vector.

For Set, the implementation classes are HashSet, TreeSet, and LinkedHashSet

Map is a different interface that does not extend the Collection interface, but it is in the Java utility package.

Collection Framework Hierarchy



Slide Note

Describe List Interface

Slide Note

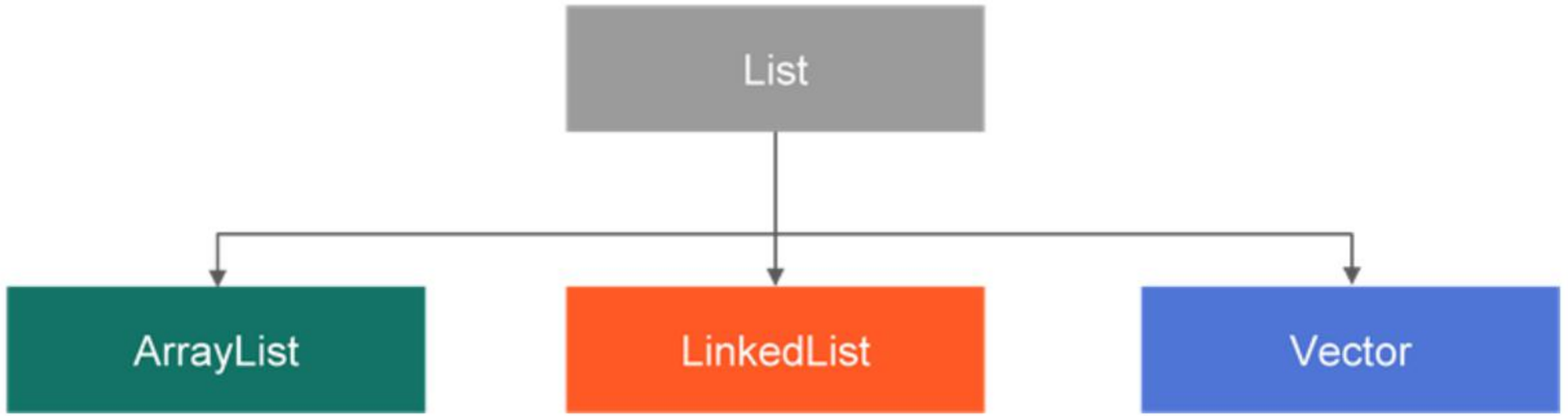
List Interface

- The List interface is used to create an ordered collection of objects.
  - The collection can be made up of predefined objects like strings, integers, floats, doubles, etc.
  - It can also be filled with user-defined objects like employees, managers, students, etc.
- An ordered collection maintains the insertion and deletion order of the elements based on position.
- Ordering gives the user precise control over the elements of the list since elements can be accessed easily based on the index position.
- A list can contain duplicate objects.

Elements of a List	Java	C++	Python	Data Structure	Angular	Python
Index Position	0	1	2	3	4	5

Slide Note

List Interface Implementation Classes



- The `ArrayList`, `LinkedList`, and `Vector` are the classes that inherit from the list interface and provide their own implementations.
- `ArrayList` enables you to create a resizable array. It is efficient at searching for an object in the list and inserting the objects at the end of the list.
- `LinkedList` enables you to create a doubly linked list.
- The `Vector` class is like the `ArrayList` except that it is synchronized.

Slide Note

Implementing ArrayList

Slide Note

- Some of the commonly used methods of the ArrayList class are:
  - boolean add(E e)
  - void clear()
  - E get(int index)
  - E remove(int index)
  - boolean remove(Object o)
  - int size()

```java
public static void main(String[] args) {
    //Creating an ArrayList object
    List list = new ArrayList();
    //calling add() and adding Integer objects
    //adding duplicate values
    list.add(10);
    list.add(10);
    //adding String objects
    list.add("Java Programming");
    list.add("Collection Framework");
    //adding Double object
    list.add(10.99);
    //adding Character object
    list.add('c');
    System.out.println("List is Ordered and it contains Duplicates elements");
    //Printing List object
    System.out.println(list);
}
```

Output

```
List is Ordered and it contains Duplicates elements
[10, 10, Java Programming, Collection Framework, 10.99, c]
```

ArrayList Set 1

- Creating an ArrayList object.
- Here List is a reference variable, and ArrayList is its implementation class.
- The boolean add(Object o) method - adds the specified element to the end of the list.
- A collection can contain heterogeneous objects, i.e., you can add objects of any type, like integers, strings, doubles, etc.
- The output produces an ordered list with duplicate data.

Slide Note

List Interface Methods

A few methods are shown below:

- `boolean add(Object o)` **- Adds the specified element to the end of the list**
- `add(int index, E element)` **- Inserts the specified element at the specified position in this list**
- `set(int index,E element)` **- Replaces the element at the specified position in this list with the specified element**
- `remove(int index)` **- Removes the element at the specified position in this list**
- `indexOf(Object o)` **- Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element**
- `get(int index)` **- Returns the element at the specified position in this list**

Slide Note

ArrayList Set 2

- The actual list

index	0	1	2	3	4	5
	10	10	Java Programming	Collection Framework	10.99	c

Size of List = 5

- Add an element 40 at the second index

```
//adding specific element at index position 2
list.add( i: 2, e: 40);
```

- The list after adding element 40

index	0	1	2	3	4	5	6
	10	10	40	Java Programming	Collection Framework	10.99	c

Size of List = 6

Slide Note

ArrayList Set 2 (Cont'd)

- New list

Size of List =6

0	1	2	3	4	5	6
10	10	40	Java Programming	Collection Framework	10.99	c

- Replaces the element at the specified position in this list with the specified element.

```
//replace element at 3rd position with String object provided
list.set(3,"Java world");
```

The list after replacing Java programming

Size of List =6

0	1	2	3	4	5	6
10	10	40	Java world	Collection Framework	10.99	c

Slide Note

ArrayList Set 3

```
//adding specific element at index position 2
list.add( i: 2,  e: 40);
//replace element at 3rd position with String object provided
list.set(3, "Java world");
//remove element form index position 6
// list.remove(6);
System.out.println("ArrayList After modification: " + list);
System.out.println("Does list contain 10 : " + list.contains(10));
System.out.println("Index of element 1.009: " + list.indexOf(1.009));
System.out.println("Get list data from index position 2: " + list.get(2));
```

```
[10, 10, Java Programming, Collection Framework, 10.99, c]
ArrayList After modification: [10, 10, 40, Java world, Collection Framework, 10.99, c]
Does list contain 10 : true
Index of element 1.009: -1
Get list data from index position 2: 40
```

Slide Note

String Object in ArrayList

Write a Java program to create a list containing string object names.
Call the following built-in methods of the list interface:

- size() - to know the size of the list.
- indexOf() - to know the index position of a given element.
- get() - to retrieve data from the given index position.
- contains() - check whether list contains the following element.

At the end, print the entire list object.

Click here for the solution.
The Intellij IDE must be used for the demonstration.

DEMO

Slide Note

```java
public static void main(String[] args) {
    //Creating an ArrayList object
    List list = new ArrayList();
    //Calling add method and adding Integer Object
    list.add(10);
    list.add(20);
    // Adding String Object
    list.add("Java Programming");
    list.add("Collection Framework");
    // Adding Double Object
    list.add(1.009);
    // Adding Character Object
    list.add('c');//Character Object
    // Printing the list object which by default will call toString method
    System.out.println("original ArrayList : " + list);
    for (Object object:list) {
        Integer number = (Integer) object;
        System.out.println(number);
    }
```

Output

```
10
20
Exception in thread "main" java.lang.ClassCastException
```

ClassCastException in Heterogenous Collection

- An Object of any datatype can be added to the list as it is heterogeneous in nature.
- The List shown in the image contains objects of all data types, like Integers, Strings, Doubles.
- So, while iterating the List using a for each loop, you need to take a container of type Object class in the for loop. Since only the Object class can store objects of all data types
- If the elements need to be displayed as an Integer for further manipulation, you will get a `ClassCastException` since there are objects of multiple data types, like string and float, which cannot be cast to the integer type.

Slide Note

Think and Tell

- How can this ClassCastException be avoided?
- Is it efficient to store heterogeneous data in a collection?
- Can a collection be used to store homogenous data to avoid the ClassCastException exception?
- Is there a rule that can be set before adding elements to the collection?
- If different types of objects are added to a collection, is it possible to alert at compile time instead of run time?

Slide Note

Introduction to Generics

Slide Note

Generics

- Generics are referred to as parameterized types. Parameterized types are important because they enable you to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter.
- The generic collections were introduced in Java 5 Version. Generic collections avoid type-casting, are type-safe, and are checked at compile time.
- Generic collections allow the data types to be passed as parameters to classes where compatibility is checked at compile-time.
- A generic class:
  - Provides type safety to code
  - Eliminates runtime exceptions
  - Provides a cleaner and easier way to write code
  - Reduces the need for casting

Slide Note

```
public class GenericClassDemo <T>{
    private T object;

    public T getObject() {
        return object;
    }
    public void setObject(T object) {
        this.object = object;
    }
}
```

Generic Class

- Generics enable you to generalize classes.
- In the declaration of a generic class, the name of the class is followed by a type parameter section.
- GenericClassDemo<T> is a generic class.
- <> The angular brackets here represent a type parameter, which can have one or more types of parameters separated by commas.
- Here, a variable named T has been added to the class definition, surrounded by the angular <> brackets. This T variable stands for “type” and can represent any type.
- This GenericClassDemo class can be used to set and get any type of object (e.g., integer, string, double, float, etc.)

Slide Note

```java
public class GenericClassDemo <T>{
    private T object;

    public T getObject() {
        return object;
    }
    public void setObject(T object) {
        this.object = object;
    }
}
```

Generic Method

- In a generic class, a method can use the type parameter of the class, which automatically makes the method generic.
- Here T represents any type, which can be a method parameter or even a method return type.
- The other advantage is the avoidance of code duplication. Without generics, they would have to rewrite the same code for different types. With generics, you do not have to do this.

Slide Note

Implementing Generics in Collections

Slide Note

Generics in Collections

- A generic List that holds Integer values is shown.
- An integer object must be specified for the generic type and not the primitive type interface.
- With generics, the compiler now enforces storing the same type of object in a collection. It is one of the many benefits of generics.

```
List<Integer> list = new ArrayList<>();
```

```
//This will give compile type error
List<int> list = new ArrayList<>();
```

Slide Note

```java
public static void main(String[] args) {
    //Creating an ArrayList of Type Integer
    List<Integer> list = new ArrayList();
    //Calling add method and adding Integer Object
    list.add(10);
    list.add(20);
    list.add(30);
    list.add(40);
    list.add(50);
    // Adding String Object will now give compile time error
    //as this list is of type Integer now
    //list.add("Java Programming");
    for (Integer number : list) {
        System.out.println(number);
    }
}
```

Output:

```
10
20
30
40
50
```

Code With Generics

- The list created will now only contain integer objects.
- Adding any other object to the list will produce a compile-time error.
- Printing elements from the list type casting is not required since all the elements in the list are of type integer.
- It prevents a `ClassCastException` at runtime.

Slide Note

Generic and Non-Generic Collections

Basic	Non-Generic	Generic
1. Syntax	List list = new ArrayList();	List<Type> list = new ArrayList<>();
2. Type safety	Can hold any type of data. So, not type-safe.	Can hold only the defined type of data. So, type safe.
3. Type casting	Individual-type casting needs to be done at every retrieval.	No type casting is needed.
4. Compile-time checking	Checked for type safety at runtime.	Checked for type safety at compile-time
5. Code reusability	Code needs to be modified for different types of objects.	Can write a method/class/interface only once and use it- for any type of object.

Slide Note

Generics Demo

Write a Java program to create two list objects.
The first list will be of type string, which will hold the names, and the other list, of type integer, will hold marks.
Use generics to achieve the above two list objects.

Make all the string objects in uppercase.
Add all the integer objects and return the sum.
Use a for-loop to iterate through the list.

Click here for the solution.
The Intellij IDE must be used for the demonstration.

DEMO

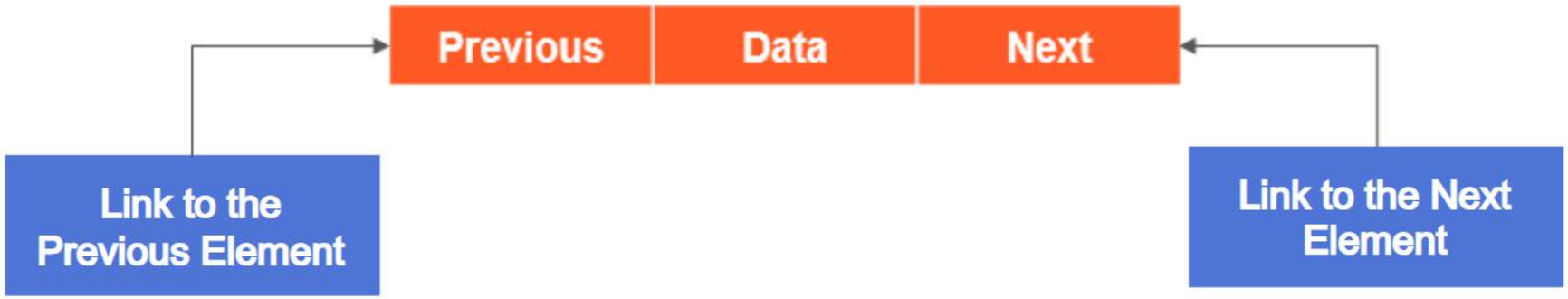
Slide Note

Implement LinkedList

Slide Note

LinkedList

- The LinkedList class of the Java collection framework provides the functionality of the linked list data structure (doubly linkedlist).
- Each element in a linked list is known as a node. It consists of 3 fields:



- Previous - Stores the address of the previous element in the list. It is null for the first element.
- Next - Stores the address of the next element in the list. It is null for the last element.
- Data - Stores the actual data or object.

Slide Note

Methods of LinkedList

```
public static void main(String[] args) {
    LinkedList<String> linkedList = new LinkedList<>();
    linkedList.add("Jenny");
    linkedList.add("Bob");
    linkedList.add("William");
    linkedList.addFirst( e: "Tim");
    linkedList.addLast( e: "Juli");
    System.out.println(linkedList);

}
```

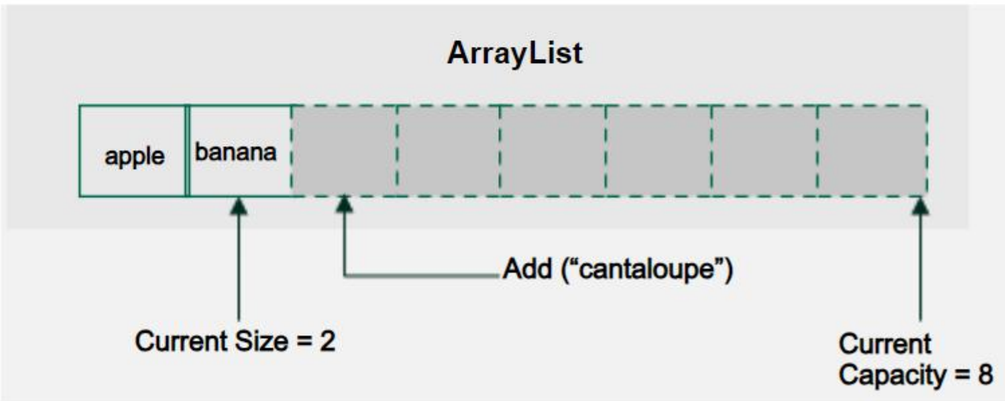
```
[Tim, Jenny, Bob, William, Juli]
```

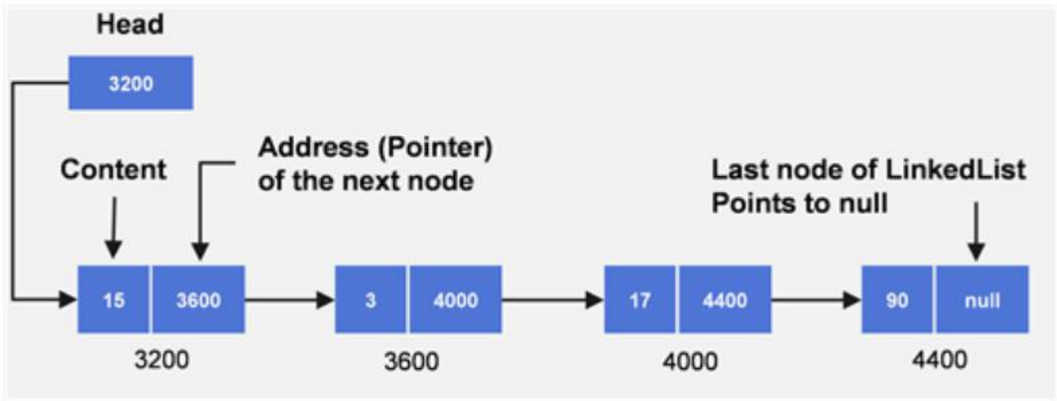
- Creating a LinkedList object of type string.
- boolean add(E e) - Appends the specified element to the end of this list.
- void addFirst(E e) - Inserts the specified element at the beginning of this list.
- void addLast(E e) - Appends the specified element to the end of this list.
- addFirst(), addLast() are specific methods of the LinkedList class only.

Slide Note

ArrayList vs. LinkedList

ArrayList	LinkedList
This class enables you to create a resizable array that is dynamically growable to store elements in it.	This class enables you to create a doubly-linked list and store elements in it.
Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array can be shrunk. The array list stores data in continuous memory locations.	LinkedList is a linear data structure.elements are not stored in contiguous locations like an ArrayList. They are linked with each other using pointers. Each element of the LinkedList has a reference (address/pointer) to the next element of the LinkedList.
When the demand is for quick insert and retrieval of data, an ArrayList should be used.	When demand for manipulation of stored data LinkedList works better.





Menu

00:37:00:00:50 37/ 50

Slide Note

Exploring Iterators in a Collection

Slide Note

A List interface has an index position, so it can be iterated through a traditional loop as well as a for-each loop.

Iterating Through List Interface

- A List interface can be iterated in four ways:
  - Traditional for loop
  - Enhanced for loop (for each)
  - Iterator interface
  - List Iterator

Traditional for loop

```
for (int i=0;i<list.size();i++){
    System.out.println(list.get(i));
}
```

`get(int index)` - Returns the element at the specified position in this list.

Enhanced for loop

```
for (Integer number : list) {
    System.out.println(number);
}
```

Here list elements will get stored in a number, and then the number is getting printed.

Slide Note

Iterator Interface Set-1

- Iterators are used in the Java collection framework to retrieve elements individually.
- The Iterator is universal for all collection objects. By using iterators, you can perform both read and remove operations:
  - `Iterator iterator = collection.iterator();`
  - Here, the collection is any collection object, and the iterator is of type Iterator interface

```
//get the iterator on list
Iterator iterator = list.iterator();
//Return true if there are more numbers of elements
while(iterator.hasNext()){
    //Return the next element
    System.out.println(iterator.next());
}
```

Menu

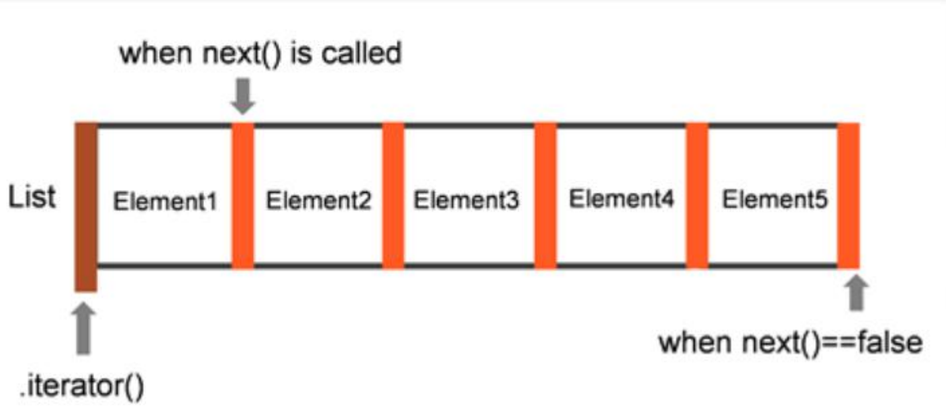
00:40:00:00:50 40/ 50

Slide Note

Iterator Interface Set-2

- Methods of Iterator interface
  - hasNext(): Returns true if the iteration has more elements.
  - next(): returns the next element in the iteration. It throws NoSuchElementException if no more elements are present.
  - remove(): Removes the next element in the iteration. This method can be called only once per call to the next.

```
    Iterator<Integer> iterator = list.iterator();
    while(iterator.hasNext()) {
        Integer integer = iterator.next();
        if(integer < 10) {
            iterator.remove();
        }
    }
    System.out.println(list);
}
```



Menu

00:41:00:00:50 41/ 50

Slide Note

ListIterator

```java
//Getting ListIterator
ListIterator<Integer> listIterator  = list.listIterator();
//Forward Direction Iteration :
while(listIterator.hasNext()){
    System.out.println(listIterator.next());
}
//Backward Direction Iteration
while(listIterator.hasPrevious()){
    System.out.println(listIterator.previous());
}
```

- ListIterator is an interface that extends the Iterator interface.
- This iterator can only be used with list-implemented classes.
- It moves in both forward and backward directions.

Slide Note

Quick Check

Which one of the following statements is true about collection classes?

1. ArrayList implements a set collection.

2. A List is a collection that can be used to implement a stack or a queue.

3. The collection classes are all stored in the java.lang package.

4.ArrayList is a dynamically growable array.

Slide Note

Quick Check: Solution

Which one of the following statements is true about collection classes?

1. ArrayList implements a set collection.

2. A List is a collection that can be used to implement a stack or a queue.

3. The collection classes are all stored in the java.lang package.

4. ArrayList is a dynamically growable array.

Slide Note

Storing User-Defined Objects in a Collection

Slide Note

User-Defined Objects

- Collections can store any kind of objects.
- An entire employee object can be stored in any collection. Here we are adding it to the list object.
- Consider that we have a list of mailing addresses that we need to store in an Array`List`.

```
public class Address {
    private String name;
    private String street;
    private String city;
    private String country;
    private int zipCode;
```

`Address` **class**

```
public class AddressImpl {
    public static void main(String[] args) {
        List<Address> addrList = new ArrayList<>();
        addrList.add(new Address( name: "Jane", street: "marble street", city: "Kolkata", country: "India", zipCode: 700027));
        addrList.add(new Address( name: "Sara", street: "hana street", city: "Bangalore", country: "India", zipCode: 560098));
        addrList.add(new Address( name: "Gary", street: "Trident enclave", city: "Hyderabad", country: "India", zipCode: 500021));
        addrList.add(new Address( name: "Holly", street: "palace pink", city: "Jaipur", country: "India", zipCode: 302001));
        addrList.add(new Address( name: "Dolly", street: "NH road", city: "Chennai", country: "India", zipCode: 600021));

    }
}
```

`AddressImpl` **class where the addresses are added to an ArrayList object.**

Slide Note

List of Students

Write a Java program to create a list of type Student.
Create a method called getAllStudents(), which will return a list of Student.
The following task needs to be done:
1. Retrieve the student whose name starts with "H".
2. Calculate the average of all the marks of all the students.
3. Remove the student from the list whose name starts with "B".
Use the Iterator to iterate through the List.

Click here for the solution.
The Intellij IDE must be used for the demonstration.

DEMO

Slide Note

Quick Check

What is the difference between the length() and size() methods of an ArrayList?

1. length() and size() return the same value.

2. length() is not a method of the ArrayList.

3. size() is not a method of the ArrayList.

4.length() returns the capacity of the ArrayList and size() returns the actual number of elements stored in the list.

Slide Note

Quick Check : Solution

What is the difference between the length() and size() methods of an ArrayList?

1. length() and size() return the same value.

2. length() is not a method of the ArrayList.

3. size() is not a method of the ArrayList.

4.length() returns the capacity of the ArrayList and size() returns the actual number of elements stored in the list.