Collection Classes

Content

- **List**
- **AbstractList,**
- **ArrayList,**
- **LinkedList,**
- **Enumeration,**
- **Vector,**
- **Properties,**
- **Introuduction to Java.util package.**

Java Collections class

- Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.
- The important points about Java Collections class are:
  - Java Collection class supports the **polymorphic algorithms** that operate on collections.
  - Java Collection class throws a **NullPointerException** if the collections or class objects provided to them are null.

- The **polymorphic algorithms** described here are pieces of reusable functionality provided by the **Java** platform. All of them come from the Collections class, and all take the form of static methods whose first argument is the collection on which the operation is to be performed.

Java List

- **List** in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.
- The List interface is found in the **java.util** package and inherits the Collection interface. It is a factory of ListIterator interface.
- Through the ListIterator, we can iterate the list in forward and backward directions.
- The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.
- List Interface declaration
- **public interface** List<E> **extends** Collection<E>

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 a list.
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.
boolean equals(Object o)	It is used to compare the specified object with the elements of a list.
int hashcode()	It is used to return the hash code value for a 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.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
<T> T[] toArray(T[] a)	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
boolean containsAll(Collection<?> c)	It returns true if the list contains all the specified element

Method	Description
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.
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.

E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int size()	It is used to return the number of elements present in the list.

- How to create List
- The ArrayList and LinkedList classes provide the implementation of List interface. Let's see the examples to create the List:

```
//Creating a List of type String using ArrayList
List<String> list=new ArrayList<String>();

//Creating a List of type Integer using ArrayList
List<Integer> list=new ArrayList<Integer>();

//Creating a List of type Book using ArrayList
List<Book> list=new ArrayList<Book>();

//Creating a List of type String using LinkedList
List<String> list=new LinkedList<String>();
```

- In short, you can create the List of any type. The ArrayList<T> and LinkedList<T> classes are used to specify the type. Here, T denotes the type.

- Java List Example

```
import java.util.*;
public class ListExample1{
public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

}
}
```

- Get and Set Element in List
- The *get() method* returns the element at the given index, whereas the *set() method* changes or replaces the element.

```
import java.util.*;
public class ListExample2{
 public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //accessing the element
 System.out.println("Returning element: "+list.get(1));//it will return the 2nd element, because index s
      tarts from 0
 //changing the element
 list.set(1,"Dates");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

 }
}
```

How to Sort List

There are various ways to sort the List, here we are going to use Collections.sort() method to sort the list element. The *java.util* package provides a utility class **Collections** which has the static method sort(). Using the **Collections.sort()** method, we can easily sort any List.

```
import java.util.*;
class SortArrayList{
 public static void main(String args[]){
  //Creating a list of fruits
  List<String> list1=new ArrayList<String>();
  list1.add("Mango");
  list1.add("Apple");
  list1.add("Banana");
  list1.add("Grapes");
  //Sorting the list
  Collections.sort(list1);
   //Traversing list through the for-each loop
  for(String fruit:list1)
    System.out.println(fruit);

 System.out.println("Sorting numbers...");
  //Creating a list of numbers
  List<Integer> list2=new ArrayList<Integer>();
  list2.add(21);
  list2.add(11);
  list2.add(51);
  list2.add(1);
  //Sorting the list
  Collections.sort(list2);
   //Traversing list through the for-each loop
  for(Integer number:list2)
    System.out.println(number);
 }

}
```

How to Sort List

There are various ways to sort the List, here we are going to use Collections.sort() method to sort the list element. The *java.util* package provides a utility class **Collections** which has the static method sort(). Using the **Collections.sort()** method, we can easily sort any List.

```
import java.util.*;
class SortArrayList{
 public static void main(String args[]){
  //Creating a list of fruits
  List<String> list1=new ArrayList<String>();
  list1.add("Mango");
  list1.add("Apple");
  list1.add("Banana");
  list1.add("Grapes");
  //Sorting the list
  Collections.sort(list1);
   //Traversing list through the for-each loop
  for(String fruit:list1)
    System.out.println(fruit);

 System.out.println("Sorting numbers...");
  //Creating a list of numbers
  List<Integer> list2=new ArrayList<Integer>();
  list2.add(21);
  list2.add(11);
  list2.add(51);
  list2.add(1);
  //Sorting the list
  Collections.sort(list2);
   //Traversing list through the for-each loop
  for(Integer number:list2)
    System.out.println(number);
 }

}
```

Java ListIterator Interface

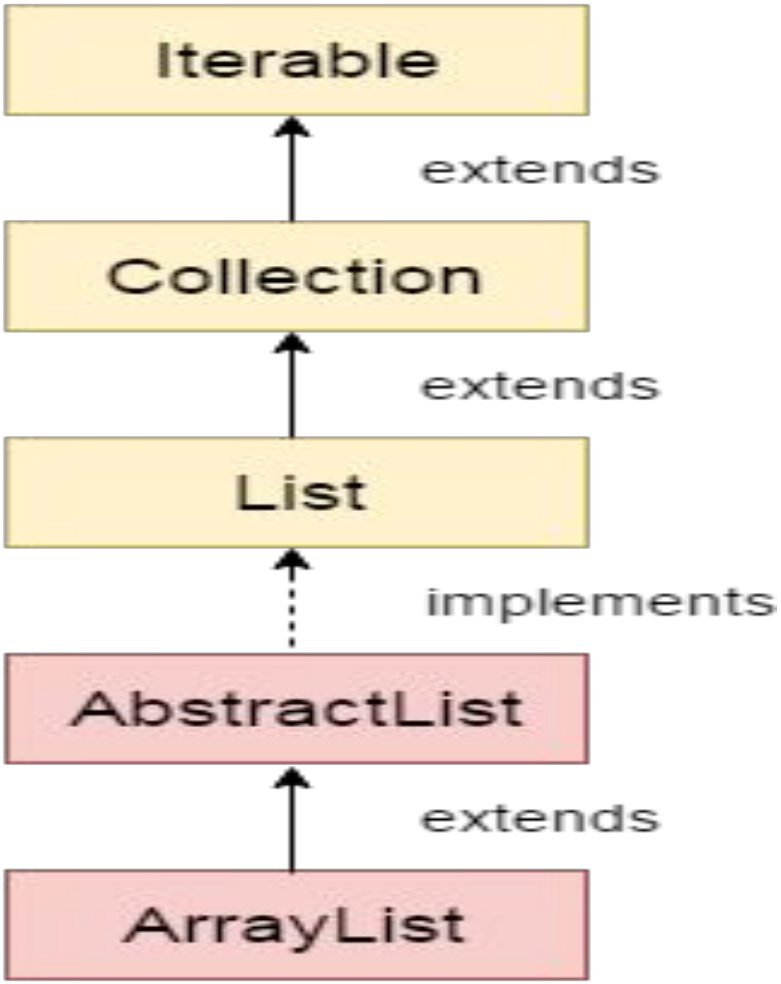
- ListIterator Interface is used to traverse the element in a backward and forward direction.
- ListIterator Interface declaration
- **public interface** ListIterator<E> **extends** Iterator<E>

Method	Description
void add(E e)	This method inserts the specified element into the list.
boolean hasNext()	This method returns true if the list iterator has more elements while traversing the list in the forward direction.
E next()	This method returns the next element in the list and advances the cursor position.
int nextIndex()	This method returns the index of the element that would be returned by a subsequent call to next()
boolean hasPrevious()	This method returns true if this list iterator has more elements while traversing the list in the reverse direction.
E previous()	This method returns the previous element in the list and moves the cursor position backward.
E previousIndex()	This method returns the index of the element that would be returned by a subsequent call to previous().
void remove()	This method removes the last element from the list that was returned by next() or previous() methods
void set(E e)	This method replaces the last element returned by next() or previous() methods with the specified element.

```
import java.util.*;
public class ListIteratorExample1{
public static void main(String args[]){
List<String> al=new ArrayList<String>();
        al.add("Amit");
        al.add("Vijay");
        al.add("Kumar");
        al.add(1,"Sachin");
        ListIterator<String> itr=al.listIterator();
        System.out.println("Traversing elements in forward direction");
        while(itr.hasNext()){

        System.out.println("index:"+itr.nextIndex()+" value:"+itr.next());
        }
  System.out.println("Traversing elements in backward direction");
        while(itr.hasPrevious()){

        System.out.println("index:"+itr.previousIndex()+" value:"+itr.previous());
        }
}
}
```

Java ArrayList

- Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the *java.util* package
- The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.
- It inherits the AbstractList class and implements List interface.
- ArrayList class declaration
- Let's see the declaration for java.util.ArrayList class.
- **public class** ArrayList<E> **extends** AbstractList<E> **implements** List <E>, RandomAccess, Cloneable, Serializable
-


Iterable
extends
Collection
extends
List
implements
AbstractList
extends
ArrayList

- The important points about Java ArrayList class are:
  - Java ArrayList class can contain duplicate elements.
  - Java ArrayList class maintains insertion order.
  - Java ArrayList class is non synchronized.
  - Java ArrayList allows random access because array works at the index basis.
  - In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

- Constructors of ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of ArrayList

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.
void ensureCapacity(int requiredCapacity)	It is used to enhance the capacity of an ArrayList instance.
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.
Iterator()	
listIterator()	
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.

<T> T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
Object clone()	It is used to return a shallow copy of an ArrayList.
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.
boolean removeIf(Predicate<? super E> filter)	It is used to remove all the elements from the list that satisfies the given predicate.
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.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
Spliterator<E> spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int size()	It is used to return the number of elements present in the list.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size

Java Non-generic Vs. Generic Collection

- Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.
- Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.
- Let's see the old non-generic example of creating java collection.
- ArrayList list=**new** ArrayList();//creating old non-generic arraylist
- Let's see the new generic example of creating java collection.
- ArrayList<String> list=**new** ArrayList<String>();//creating new gene ric arraylist
- In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Java ArrayList Example

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arra
     ylist
      list.add("Mango");//Adding object in arraylist
      list.add("Apple");
      list.add("Banana");
      list.add("Grapes");
      //Printing the arraylist object
      System.out.println(list);
 }
}
```

Iterating ArrayList using Iterator

- Let's see an example to traverse ArrayList elements using the Iterator interface.

```
import java.util.*;
public class ArrayListExample2{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through Iterator
  Iterator itr=list.iterator();//getting the Iterator
  while(itr.hasNext()){//check if iterator has the elements
   System.out.println(itr.next());//printing the element and move to next
  }
 }
}
```

Iterating ArrayList using For-each loop

Let's see an example to traverse the ArrayList elements using the for-each loop

```
import java.util.*;
public class ArrayListExample3{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through for-each loop
  for(String fruit:list)
    System.out.println(fruit);

 }
}
```

How to convert List to Array

- We can convert the List to Array by calling the list.toArray() method. Let's see a simple example to convert list elements into array.

```
import java.util.*;
public class ListToArrayExample{
public static void main(String args[]){
 List<String> fruitList = new ArrayList<>();
 fruitList.add("Mango");
 fruitList.add("Banana");
 fruitList.add("Apple");
 fruitList.add("Strawberry");
 //Converting ArrayList to Array
 String[] array = fruitList.toArray(new String[fruitList.size()]);
 System.out.println("Printing Array: "+Arrays.toString(array));
 System.out.println("Printing List: "+fruitList);
}
}
```

- How to convert Array to List
- We can convert the Array to List by traversing the array and adding the element in list one by one using list.add() method. Let's see a simple example to convert array elements into List.

```
import java.util.*;
public class ArrayToListExample{
public static void main(String args[]){
//Creating Array
String[] array={"Java","Python","PHP","C++"};
System.out.println("Printing Array: "+Arrays.toString(array));
//Converting Array to List
List<String> list=new ArrayList<String>();
for(String lang:array){
list.add(lang);
}
System.out.println("Printing List: "+list);

}
}
```

Ways to iterate the elements of the collection in Java

- There are various ways to traverse the collection elements:
  - By Iterator interface.
  - By for-each loop.
  - By ListIterator interface.
  - By for loop.
  - By forEach() method.
  - By forEachRemaining() method.

```
import java.util.*;
class ArrayList4{
 public static void main(String args[]){
   ArrayList<String> list=new ArrayList<String>();//Creating arraylist
        list.add(“Rahul");//Adding object in arraylist
        list.add("Vijay");
        list.add("Rahul");
        list.add("Ajay");

        System.out.println("Traversing list through List Iterator:");
        //Here, element iterates in reverse order
          ListIterator<String> list1=list.listIterator(list.size());
          while(list1.hasPrevious())
          {
             String str=list1.previous();
             System.out.println(str);
          }
```

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

      System.out.println("Traversing list through forEach() method:");
      //The forEach() method is a new feature, introduced in Java 8.
         list.forEach(a->{ //Here, we are using lambda expression
            System.out.println(a);
          });

         System.out.println("Traversing list through forEachRemaining() method:");
          Iterator<String> itr=list.iterator();
          itr.forEachRemaining(a-> //Here, we are using lambda expression
          {
         System.out.println(a);
          });
 }
}
```

User-defined class objects in Java ArrayList

```
class Student{
  int rollno;
  String name;
  int age;
  Student(int rollno,String n
     ame,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}
```

```
import java.util.*;
 class ArrayList5{
 public static void main(String args[]){
  //Creating user-defined class objects
  Student s1=new Student(101,"Sonoo",23);
  Student s2=new Student(102,"Ravi",21);
  Student s2=new Student(103,"Hanumat",25);

  //creating arraylist
  ArrayList<Student> al=new ArrayList<Student
>();
  al.add(s1);//adding Student class object
  al.add(s2);
  al.add(s3);
  //Getting Iterator
  Iterator itr=al.iterator();
  //traversing elements of ArrayList object
  while(itr.hasNext()){
    Student st=(Student)itr.next();
    System.out.println(st.rollno+" "+st.name+" "
+st.age);
  }
 }
}
```

Java ArrayList Serialization and Deserialization Example

```
class ArrayList6 {

    public static void main(String [] args)
    {
      ArrayList<String> al=new ArrayList<String>();
      al.add("Ravi");
      al.add("Vijay");
      al.add("Ajay");

      try
      {
        //Serialization
        FileOutputStream fos=new FileOutputStream(
"file");
        ObjectOutputStream oos=new ObjectOutputSt
ream(fos);
        oos.writeObject(al);
        fos.close();
        oos.close();

 //Deserialization
          FileInputStream fis=new FileInputStream
("file");
          ObjectInputStream ois=new ObjectInput
Stream(fis);
         ArrayList  list=(ArrayList)ois.readObject();

          System.out.println(list);
        }catch(Exception e)
        {
          System.out.println(e);
        }
      }
    }
```

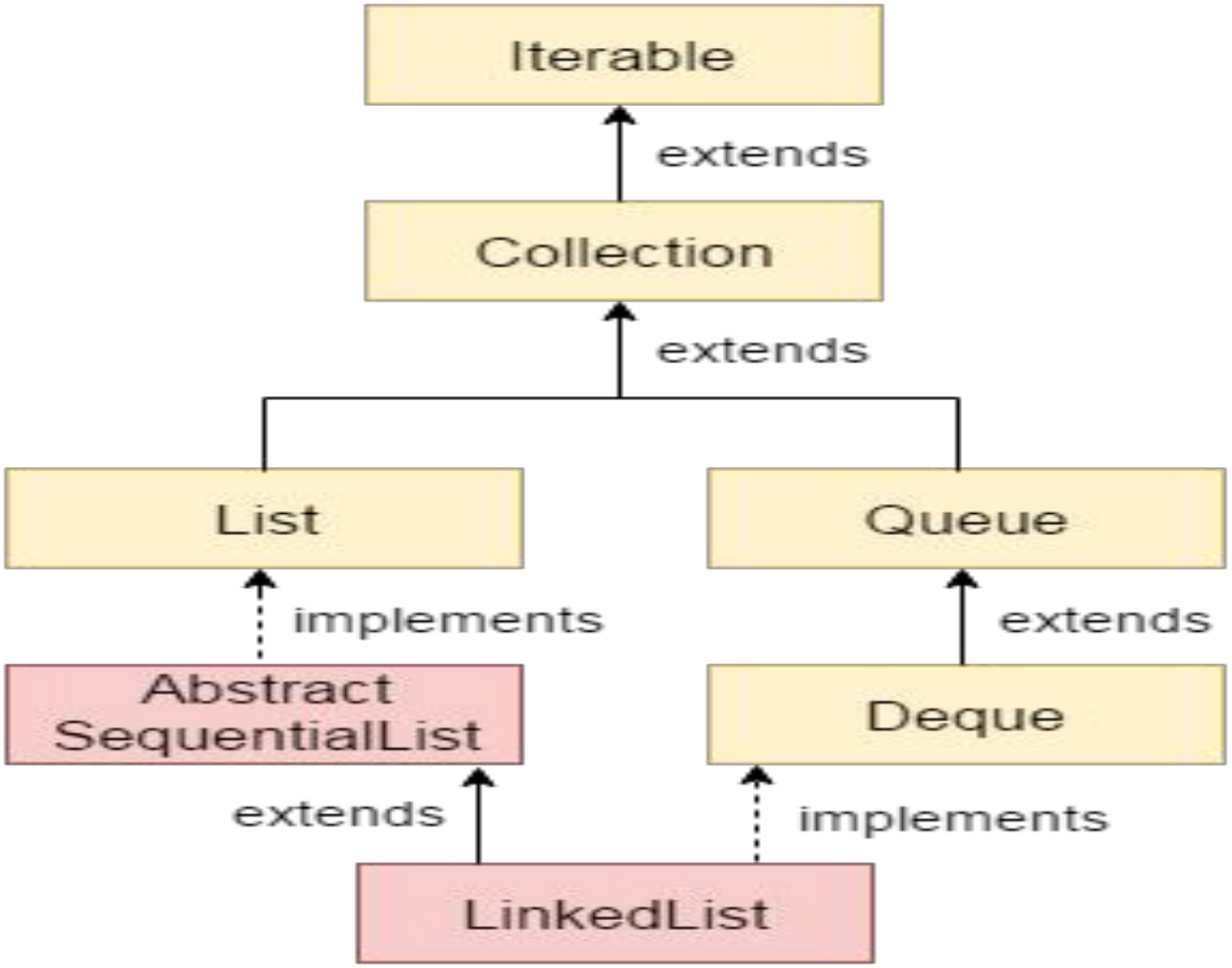
- Java ArrayList example of retainAll() method

```
import java.util.*;
class ArrayList9{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add(“Rahul");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add(“Rahul");
  al2.add("Hanumat");
  al.retainAll(al2);
  System.out.println("iterating the elements after retaining the elements of al2");
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Java LinkedList class

- Java 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 interfaces.
- The important points about Java LinkedList are:
  - Java LinkedList class can contain duplicate elements.
  - Java LinkedList class maintains insertion order.
  - Java LinkedList class is non synchronized.
  - In Java LinkedList class, manipulation is fast because no shifting needs to occur.
  - Java LinkedList class can be used as a list, stack or queue.

Hierarchy of LinkedList class



- LinkedList class declaration
- Let's see the declaration for java.util.LinkedList class.
- **public class** LinkedList<E> **extends** AbstractSequentialList<E > **implements** List<E>, Deque<E>, Cloneable, Serializable
- Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection<? extends E> c)	It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

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(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.

Object clone()	It is used to return a shallow copy of an ArrayList.
boolean contains(Object o)	It is used to return true if a list contains a specified element.
Iterator<E> descendingIterator()	It is used to return an iterator over the elements in a deque in reverse sequential order.
E element()	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
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.
ListIterator<E> listIterator(int index)	It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.

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.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.
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.

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.
Object[] toArray()	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
<T> T[] toArray(T[] a)	It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
int size()	It is used to return the number of elements in a list.

Java LinkedList example to add elements

```
public class LinkedList2{
 public static void main(String args[]){
 LinkedList<String> ll=new LinkedList<String>();
        System.out.println("Initial list of elements: "+ll);
        ll.add(“Rahul");
        ll.add("Vijay");
        ll.add("Ajay");
        System.out.println("After invoking add(E e) method: "+ll);
        //Adding an element at the specific position
        ll.add(1, "Gaurav");
        System.out.println("After invoking add(int index, E element) method: "+
    ll);
        LinkedList<String> ll2=new LinkedList<String>();
        ll2.add("Sonoo");
        ll2.add("Hanumat");
        //Adding second list elements to the first list
        ll.addAll(ll2);
```

```
    System.out.println("After invoking addAll(Collection<? extends E> c) metho
      d: "+ll);
        LinkedList<String> ll3=new LinkedList<String>();
        ll3.add("John");
        ll3.add("Rahul");
        //Adding second list elements to the first list at specific position
        ll.addAll(1, ll3);
        System.out.println("After invoking addAll(int index, Collection<? extend
      s E> c) method: "+ll);
        //Adding an element at the first position
        ll.addFirst("Lokesh");
        System.out.println("After invoking addFirst(E e) method: "+ll);
        //Adding an element at the last position
        ll.addLast("Harsh");
        System.out.println("After invoking addLast(E e) method: "+ll);

 }
}
```

Java LinkedList Example to reverse a list of elements

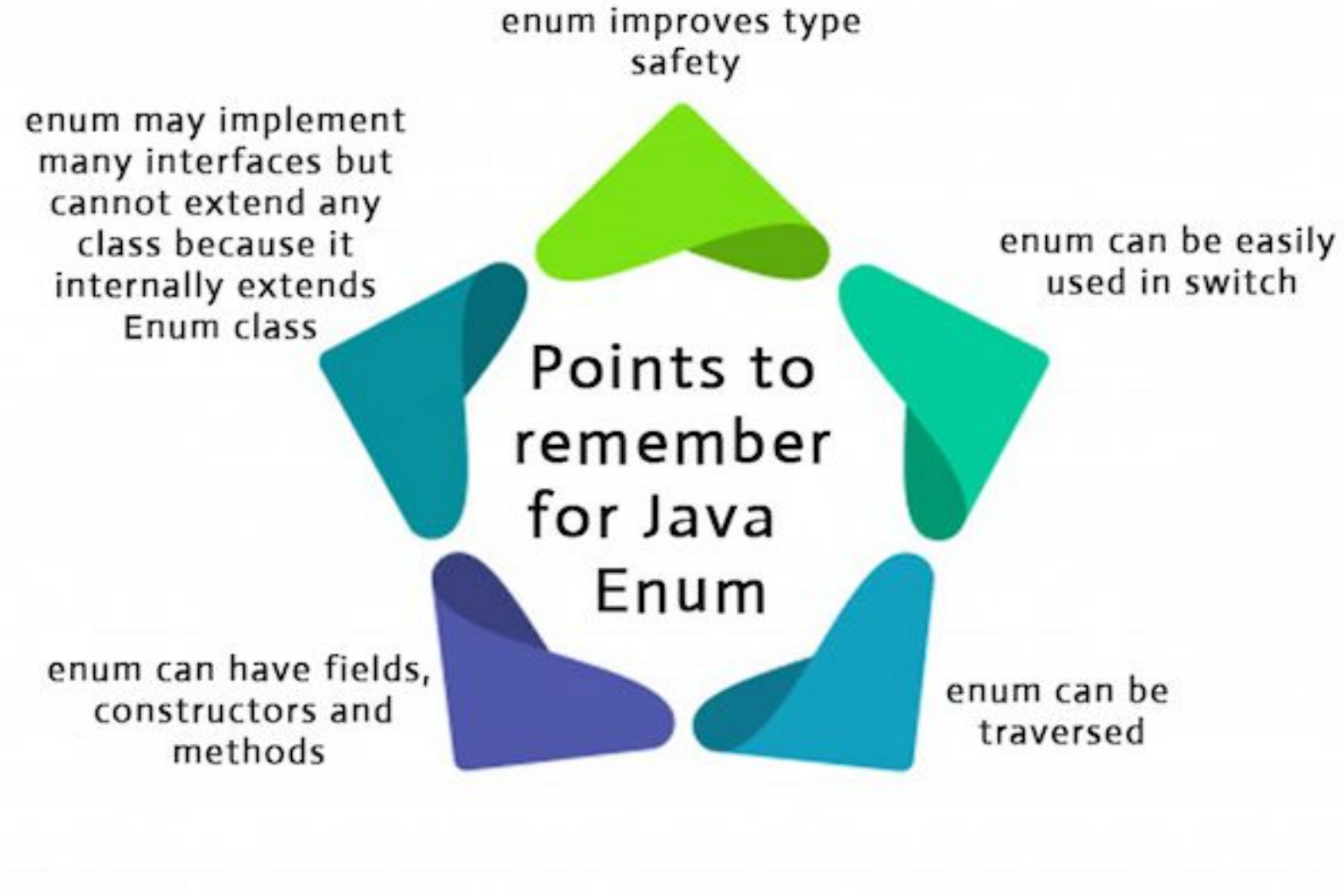
```
public class LinkedList4{
public static void main(String args[]){

 LinkedList<String> ll=new LinkedList<String>();
        ll.add("Rahul");
        ll.add("Vijay");
        ll.add("Ajay");
        //Traversing the list of elements in reverse order
        Iterator i=ll.descendingIterator();
        while(i.hasNext())
        {
           System.out.println(i.next());
        }

 }
}
```

Difference between ArrayList and LinkedList

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) 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.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) 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.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Enumartion

- The **Enum in Java** is a data type which contains a fixed set of constants.
- It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc.
- According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.
- Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.
- Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.


enum improves type safety
enum may implement many interfaces but cannot extend any class because it internally extends Enum class
enum can be easily used in switch
Points to remember for Java Enum
enum can have fields, constructors and methods
enum can be traversed

Simple Example of Java Enum

```
class EnumExample1{
//defining the enum inside the class
public enum Season { WINTER, SPRING, SUMMER, F
    ALL }
//main method
public static void main(String[] args) {
//traversing the enum
for (Season s : Season.values())
System.out.println(s);
}}
```

```
class EnumExample1{
//defining enum within class
public enum Season { WINTER, SPRING, SUMMER, FALL }
//creating the main method
public static void main(String[] args) {
//printing all enum
for (Season s : Season.values()){
System.out.println(s);
}
System.out.println("Value of WINTER is: "+Season.valueOf("WINTER"
    ));
System.out.println("Index of WINTER is: "+Season.valueOf("WINTER")
    .ordinal());
System.out.println("Index of SUMMER is: "+Season.valueOf("SUMME
    R").ordinal());

}}
```

- What is the purpose of the values() method in the enum?
  - The Java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.
- What is the purpose of the valueOf() method in the enum?
  - The Java compiler internally adds the valueOf() method when it creates an enum. The valueOf() method returns the value of given constant enum.
- What is the purpose of the ordinal() method in the enum?
  - The Java compiler internally adds the ordinal() method when it creates an enum. The ordinal() method returns the index of the enum value.

Java Enum Example: Defined inside class

```
class EnumExample3{
enum Season { WINTER, SPRING, SUMMER, FALL
   ; }
public static void main(String[] args) {
Season s=Season.WINTER;//enum type is requir
   ed to access WINTER
System.out.println(s);
}}
```

- Initializing specific values to the enum constants
- The enum constants have an initial value which starts from 0, 1, 2, 3, and so on. But, we can initialize the specific value to the enum constants by defining fields and constructors. As specified earlier, Enum can have fields, constructors, and methods.
- Example of specifying initial value to the enum constants

```
class EnumExample4{
enum Season{
WINTER(5), SPRING(10), SUMMER(15), FALL(20);

private int value;
private Season(int value){
this.value=value;
}
}
public static void main(String args[]){
for (Season s : Season.values())
System.out.println(s+" "+s.value);

}}
```

- Constructor of enum type is private. If you don't declare private compiler internally creates private constructor.

```
enum Season{
WINTER(10),SUMMER(20);
private int value;
Season(int value){
this.value=value;
}
}
```

- Internal code generated by the compiler for the above example of enum type

```
final class Season extends Enum
{
   public static Season[] values()
   {
      return (Season[])$VALUES.clone();
   }
   public static Season valueOf(String s)
   {
      return (Season)Enum.valueOf(Season, s);
   }
   private Season(String s, int i, int j)
   {
      super(s, i);
      value = j;
   }
   public static final Season WINTER;
   public static final Season SUMMER;
   private int value;
   private static final Season $VALUES[];
   static
   {
      WINTER = new Season("WINTER", 0, 10);
      SUMMER = new Season("SUMMER", 1, 20);
      $VALUES = (new Season[] {
         WINTER, SUMMER
      });
   }
}
```

Java Enum in a switch statement

```
class EnumExample5{
enum Day{ SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}

public static void main(String args[]){
Day day=Day.MONDAY;

switch(day){
case SUNDAY:
 System.out.println("sunday");
 break;
case MONDAY:
 System.out.println("monday");
 break;
default:
System.out.println("other day");
}
}}
```

Java Vector

- **Vector** is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.
- It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the *List* interface, so we can use all the methods of List interface here.
- It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.
- The Iterators returned by the Vector class are *fail-fast*. In case of concurrent modification, it fails and throws the ConcurrentModificationException.
- It is similar to the ArrayList, but with two differences-
  - Vector is synchronized.
  - Java Vector contains many legacy methods that are not the part of a collections framework.

Vector Constructor

SN	Constructor	Description
1)	vector()	It constructs an empty vector with the default size as 10.
2)	vector(int initialCapacity)	It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
3)	vector(int initialCapacity, int capacityIncrement)	It constructs an empty vector with the specified initial capacity and capacity increment.
4)	Vector(Collection<? extends E> c)	It constructs a vector that contains the elements of a collection c.

SN	Method	Description
1)	add()	It is used to append the specified element in the given vector.
2)	addAll()	It is used to append all of the elements in the specified collection to the end of this Vector.
3)	addElement()	It is used to append the specified component to the end of this vector. It increases the vector size by one.
4)	capacity()	It is used to get the current capacity of this vector.
5)	clear()	It is used to delete all of the elements from this vector.
6)	clone()	It returns a clone of this vector.
7)	contains()	It returns true if the vector contains the specified element.
8)	containsAll()	It returns true if the vector contains all of the elements in the specified collection.

9)	copyInto()	It is used to copy the components of the vector into the specified array.
10)	elementAt()	It is used to get the component at the specified index.
11)	elements()	It returns an enumeration of the components of a vector.
12)	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.
13)	equals()	It is used to compare the specified object with the vector for equality.
14)	firstElement()	It is used to get the first component of the vector.
15)	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.
16)	get()	It is used to get an element at the specified position in the vector.
17)	hashCode()	It is used to get the hash code value of a vector.

8)	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.
19)	insertElementAt()	It is used to insert the specified object as a component in the given vector at the specified index.
20)	isEmpty()	It is used to check if this vector has no components.
21)	iterator()	It is used to get an iterator over the elements in the list in proper sequence.
22)	lastElement()	It is used to get the last component of the vector.
23)	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.
24)	listIterator()	It is used to get a list iterator over the elements in the list in proper sequence.
25)	remove()	It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged.
26)	removeAll()	It is used to delete all the elements from the vector that are present in the specified collection.

27)	removeAllElements()	It is used to remove all elements from the vector and set the size of the vector to zero.
28)	removeElement()	It is used to remove the first (lowest-indexed) occurrence of the argument from the vector.
29)	removeElementAt()	It is used to delete the component at the specified index.
30)	removeIf()	It is used to remove all of the elements of the collection that satisfy the given predicate.
31)	removeRange()	It is used to delete all of the elements from the vector whose index is between fromIndex, inclusive and toIndex, exclusive.
32)	replaceAll()	It is used to replace each element of the list with the result of applying the operator to that element.
33)	retainAll()	It is used to retain only that element in the vector which is contained in the specified collection.
34)	set()	It is used to replace the element at the specified position in the vector with the specified element.
35)	setElementAt()	It is used to set the component at the specified index of the vector to the specified object.
36)	setSize()	It is used to set the size of the given vector.
37)	size()	It is used to get the number of components in the given vector.
38)	sort()	It is used to sort the list according to the order induced by the specified Comparator.

39)	spliterator()	It is used to create a late-binding and fail-fast Spliterator over the elements in the list.
40)	subList()	It is used to get a view of the portion of the list between fromIndex, inclusive, and toIndex, exclusive.
41)	toArray()	It is used to get an array containing all of the elements in this vector in correct order.
42)	toString()	It is used to get a string representation of the vector.
43)	trimToSize()	It is used to trim the capacity of the vector to the vector's current size.

```
import java.util.*;
public class VectorExample {
      public static void main(String args[]) {
        //Create a vector
        Vector<String> vec = new Vector<String>();
        //Adding elements using add() method of List
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        //Adding elements using addElement() method of Vector
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");

        System.out.println("Elements are: "+vec);
      }
}
```

Properties class in Java

- The **properties** object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.
- It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.
- An Advantage of the properties file
  - **Recompilation is not required if the information is changed from a properties file:** If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.

Constructors of Properties class

Method	Description
Properties()	It creates an empty property list with no default values.
Properties(Properties defaults)	It creates an empty property list with the specified defaults.

Method	Description
public void load(Reader r)	It loads data from the Reader object.
public void load(InputStream is)	It loads data from the InputStream object
public void loadFromXML(InputStream in)	It is used to load all of the properties represented by the XML document on the specified input stream into this properties table.
public String getProperty(String key)	It returns value based on the key.
public String getProperty(String key, String defaultValue)	It searches for the property with the specified key.
public void setProperty(String key, String value)	It calls the put m

public void list(PrintStream out)	It is used to print the property list out to the specified output stream.
public void list(PrintWriter out))	It is used to print the property list out to the specified output stream.
public Enumeration<?> propertyNames())	It returns an enumeration of all the keys from the property list.
public Set<String> stringPropertyNames()	It returns a set of keys in from property list where the key and its corresponding value are strings.
public void store(Writer w, String comment)	It writes the properties in the writer object.
public void store(OutputStream os, String comment)	It writes the properties in the OutputStream object.
public void storeToXML(OutputStream os, String comment)	It writes the properties in the writer object for generating XML document.
public void storeToXML(Writer w, String comment, String encoding)	It writes the properties in the writer object for generating XML document with the specified encoding.

- Example of Properties class to get information from the properties file
- To get information from the properties file, create the properties file first.
  - **db.properties**
  - user=system
  - password=oracle
- Now, let's create the java class to read the data from the properties file.
- **Test.java**

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{
   FileReader reader=new FileReader("db.properties");

   Properties p=new Properties();
   p.load(reader);

   System.out.println(p.getProperty("user"));
   System.out.println(p.getProperty("password"));
}
}
```

Example of Properties class to get all the system properties

```
public class Test {
public static void main(String[] args)throws Exception{

Properties p=System.getProperties();
Set set=p.entrySet();

Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry entry=(Map.Entry)itr.next();
System.out.println(entry.getKey()+" = "+entry.getValue());
}

}
}
```

- Example of Properties class to create the properties file

- **Test.java**

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{

Properties p=new Properties();
p.setProperty("name",“Vaidehi Patel");
p.setProperty("email",“vaidehipatel.ce@indusuni.ac.in");

p.store(new FileWriter("info.properties")," Properties Example");

}
}
```