Example -4 : Create LinkedList to add Item object and sort the list with specified keys and order.

File : **Item.java**

```
//  The Comparable interface needs to be used to sort on a single property of the object.
//  To sort with multiple properties, you need Comparator.

public class Item implements Comparable<Item> {

    //  We will use this item class to create item object for storing in Linked list

    private String description;       // Item description
    private int units;                // Units on hand

    public Item()  {    description = "";   units = 0;    }
    public Item(String d, int u)
    {
        description = d;    units = u;
    }
    public void setDescription(String d)  { description = d; }
    public void setUnits(int u)           { units = u; }
    public String getDescription()        { return description;  }
    public int getUnits()                 { return units; }

    //  Linklist Example will use this overridden equal for comparing the items
    public boolean equals(Object o )
    {
        return ((Item)o).getDescription().equals(this.getDescription());
    }

    //  Following code will help to sort the Item object based on description and unit
    //  It contains a static ItemDescComparator method to compare the “Description”.
    //  Now the Item object is able to sort with either “units” or “description” property.
    //  Example :  linkedlist.sort(items, Item.ItemDescComparator)
    public int compareTo(Item comItm)
    {
        int comUnit = comItm.getUnits();
        //ascending order
            return this.units - comUnit;
        //descending order
            //return comUnit - this.units;
    }
    //  This coprator will be used to sort description (ascending) wise only
```

```
  public static Comparator<Item> ItemDescComparator
                = new Comparator<Item>()    {
    public int compare(Item itm1, Item itm2)
    {
         String desc1 = itm1.getDescription().toUpperCase();
         String desc2 = itm2.getDescription().toUpperCase();

         //ascending order
         return desc1.compareTo(desc2);
      //descending order
         //return desc2.compareTo(desc1);
       }
  };
```

We are using anonymous class to override the compare() method of **Comparato** interface. Here new is not really creating any interface, it is creating an object **ItemDescComparator** which will return a Comparator. In this anonymous class we are overriding the compare() method.

```
  //  This coprator will be used to sort description (ascending) + units (ascending) wise
  public static Comparator<Item> ItemDescUnitComparator
                = new Comparator<Item>()    {
    public int compare(Item itm1, Item itm2)
    {
      String desc1 = itm1.getDescription().toUpperCase();
         String desc2 = itm2.getDescription().toUpperCase();

      int value1 = desc1.compareTo(desc2);
      if (value1 == 0)
      {
        int value2 = itm1.getUnits() - itm2.getUnits();
        return value2;
      }
      return value1;
    }
  };
}
```

Driver routine for above example :

```
private static void driverItemListSort()
  {
    LinkedList<Item> lnkList = new LinkedList<Item>();
    LinkListManagement  lx = new LinkListManagement();
    lx.AddInList(lnkList, "Desktop Computer", 3, "A", 0);
    lx.AddInList(lnkList, "Laptop Computer", 8,"A", 0);
    lx.AddInList(lnkList, "Hard Disk", 10, "A", 0);
    lx.AddInList(lnkList, "Pen drive", 25, "A", 0);
    lx.AddInList(lnkList, "Memory Chips", 190, "F", 0);
    lx.AddInList(lnkList, "Keboard", 50, "L", 0);
    lx.AddInList(lnkList, "Mouse", 100, "A", 0);
```

```
      lx.AddInList(lnkList, "Light Pen", 20, "I", 1);
      lx.AddInList(lnkList, "Projector", 50, "I", 4);
      lx.AddInList(lnkList, "Projector", 20, "A", 0);

      System.out.println("Unsorted Item List : ");
      lx.PrintItmList(lnkList);

      //  Sort the list by Description
      //  Sort based on description (ascending). It will use Compare method of
      //  ItemDescComparator comparator defined in Item class
      Collections.sort(lnkList, Item.ItemDescComparator);       //  For Java 8
      //Collections.sort(lnkList, new Item.ItemDescComparator());   //  For Java 7
      System.out.println("Sort on Description ascending order : ");
      lx.PrintItmList(lnkList);

      //  Sort the list by unit
      //  Sort based on unit. It will use CompareTo method defined in Item class
      Collections.sort(lnkList);
      System.out.println("Sort on Unit ascending order : ");
      lx.PrintItmList(lnkList);

      //  Sort the list by description + unit
      //  Sort based on description (ascending) + Units. It will use Compare method of
      //  ItemDescUnitComparator comparator defined in Item class
      Collections.sort(lnkList, Item.ItemDescUnitComparator);       //  For Java 8.0
      //Collections.sort(lnkList, new Item.ItemDescUnitComparator());    //  For Java 7.0
      System.out.println("Sort on Description + Unit ascending order : ");
      lx.PrintItmList(lnkList);
   }
```

Example -5 : Create Item database (memory) using LinkedList generic collection and perform user defined sorting, reversing etc. using collections methods. We use the same Item class.

File : LinkListManagement.java

```
package javaapp1;

import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
public class LinkListManagement {
   //  Add elements at beginning and end or insert after a particular index
   public void AddInList(LinkedList lList, String desc, int unt, String opn, int pos)
   {
      Item itm = new Item(desc,unt);
   switch (opn) {
      case "A" :          //  Append
            lList.add(itm);           break;
      case "F" :          //  Add First
            lList.addFirst(itm);       break;
      case "L" :          //  Add Last
            lList.addLast(itm);        break;
      case "I" :          //  Insert after a particular index
           // It inserts specified element at specified index in the LinkedList by
           // shifting current elements and subsequent elements to the right.
            lList.add(pos, itm);       break;
      default  :
   }
}
//  Print all Items stored in Linked list
public void PrintItmList(LinkedList itmList)
{
   int idx = 0;
   for (Object obj : itmList)
   {
      Item itm = (Item)obj;
      System.out.println("Item Index " + idx + " : " + itm.getDescription() + " , " + itm.getUnits() );
      idx++;
   }
}
//  Search an Item stored in list
public void SearchItm(LinkedList itmList, Item itm)
{
   Item itFnd = null;

   //  This method returns true if LinkedList contains a particular item, false otherwise.
   //  All search method like contains, indexof etc of Linklist will use the overridden
   //  equal method in Item class while searching objects for equality.
   //  This equal method will only compare the description to find the item in the list
   boolean blnElement = itmList.contains(itm);
   if(blnElement)
   {
      itFnd =(Item) itmList.get(itmList.indexOf(itm));      //  Get the item in the list
      System.out.println("LinkedList contains " + itFnd.getDescription() + " Unit : " + itFnd.getUnits());
   }
   else
   {
      System.out.println("LinkedList does not contain " + itm.getDescription());
   }
```

```
    //  To search first occurrence of an element of LinkedList, use
    //  int indexOf(Object element) method. This method returns index of first
    //  occurrence of element if found in the LinkedList. It returns -1 if element not found.
    int index = 0;
    index = itmList.indexOf(itm);
    if(index != -1)
    {
      itFnd =(Item) itmList.get(index);  //  Get the item in the specific Index of the list
      System.out.println("First occurrence of item " + itFnd.getDescription() + " Unit : " +
          itFnd.getUnits() + " in LinkedList is at index : " + index);
    }
    else
    {
     System.out.println("LinkedList does not contain the item");
    }

    //  To search last occurrence of an element of LinkedList, use lastIndexOf(Object element) method.
    //  This method returns index of last occurrence of element if found in the LinkedList.
    //  It returns -1 if element not found.
    index = itmList.lastIndexOf(itm);
    if(index != -1)
    {
      itFnd =(Item) itmList.get(index);  //  Get the item in the specific Index of the list
      System.out.println("Last occurrence of of item " + itFnd.getDescription() + " Unit : "
           + itFnd.getUnits() +  " in LinkedList is at index : " + index);
    }
    else
    {
      System.out.println("LinkedList does not contain the item");
    }
  }
  //  Get a sublist from original list
  public void GetSubList(LinkedList itmList, int stIdx, int enIdx)
  {
    //  To get a sublist from Java LinkedList we use subList(int start, int end) method.
    //  This method returns portion of list containing element from start index
    //  inclusive to end index exclusive.
    List lst = itmList.subList(stIdx,enIdx);
    System.out.println("Sublist contains : " );
    for (Object obj : lst)
    {
      Item itm = (Item)obj;
      System.out.println(itm.getDescription() + " Unit : " + itm.getUnits());
    }
    //  Please note that sublist is backed by the original list, so any changes
    //  made to sublist will also be reflected back to original LinkedList
    //  For example we remove Item at Index 2 from sublist
    lst.remove(2);
    // System.out.println("Sublist now contains : " + lst);
    System.out.println("After removal of 2nd Index Original List contains : ");
    PrintItmList(itmList);
}
//  Replace unit of first occurance of an Item object
public void ReplaceUnit(LinkedList itmList, Item itm)
{
    //  To replace an element of LinkedList at specified index, use
    //  Object set(int index, Object element) method.
    //  It replaces specified element at specified index in the LinkedList and
    //  returns the element previously at the specified index.
    int index = itmList.indexOf(itm);
    if(index != -1)
    {
        itmList.set(index, itm);
        System.out.println("After Replacing the List contains : ");
        PrintItmList(itmList);
    }
}
//  Change the Description of all occurance of an Item
public void ReplaceAllDesc(LinkedList itmList, String exDesc, String newDesc)
{
    //  To get an ListIterator object of LinkedList, use ListIterator listIterator() method.
    //  Iterating through elements in forward direction...
    ListIterator itr = itmList.listIterator();
    while(itr.hasNext())
    {
        Item exItm = (Item)itr.next();
        if ( exDesc.equals(exItm.getDescription()) )
        {
            exItm.setDescription(newDesc);
            itr.set(exItm);
        }
    }
    PrintItmList(itmList);
}
//  Print all element of LinkedList in forward and reverse direction using ListIterato
public void PrintAllFrdBck(LinkedList itmList)
{
    //  To get an ListIterator object of LinkedList, use ListIterator listIterator() method.
    //  Iterating through elements in forward direction...
    ListIterator itr = itmList.listIterator();      Item itm = null;
    System.out.println("Iterating through elements of Java LinkedList using " +
                    "ListIterator in forward direction...");
    while(itr.hasNext())
    {
        itm = (Item)itr.next();
        System.out.println(itm.getDescription() + " , " + itm.getUnits() );
    }
    System.out.println("Iterating through elements of Java LinkedList using " +
                    "ListIterator in reverse direction...");
    while(itr.hasPrevious())
    {
        itm = (Item)itr.previous();
        System.out.println(itm.getDescription() + " , " + itm.getUnits() );
    }
 }
//  Remove an Item or range of elements from Link List
public void RemoveItem(LinkedList itmList, Item itm, int stIdx, int enIdx)
{
    if ( itm != null)      //  Remove a particular item
    {
        //  To remove a specified element from Java LinkedList, use boolean remove(Object obj)
        //  method. This method removes the first occurrence of the specified element and returns true
        //  if specified element in list. If specified element not exist, list remains unchanged.

        boolean isRemoved = itmList.remove(itm);
        System.out.println("Is item removed from LinkedList ? :" + isRemoved);
        System.out.println("LinkedList now contains : " );
        PrintItmList(itmList);
    }
    else
    {
        if ( enIdx != 0 && enIdx >= stIdx) {        //  Remove a range of index

            //  Removing range of elements is not directly supported. However, it can be done by using
            //  subList and clear methods. remove elements from index stIdx(inclusive) to enIdx(exclusive)

            itmList.subList(stIdx, enIdx).clear();
            System.out.println("Range of elements removed from LinkedList.... " +
                                "LinkedList now contains : ");
            PrintItmList(itmList);
        }
```

```
         else  {    //  Remove a particular Index stated in stIdx

               //  To remove an element at specified index of LinkedList, use Object remove(int index)
               //  method, which removes an element from specified index and shifts subsequent
               //  elements to the left. It returns an element previously at the specified index.

               Object obj = itmList.remove(stIdx);    Item rmItm = (Item)obj;
               System.out.println(rmItm.getDescription() + " , " + rmItm.getUnits() +
                              " has been removed from LinkedList");
               PrintItmList(itmList);
         }
      }
   }
}
```

File : **JavaApp1.java**

```
private static void driverLinkListItem()
   {
      //  Create a Link List and Perform Various operations
      //  If we use LinkedList lnkList = new LinkedList(); the warning like unchecked or unsafe
      //  oreration in souce file will come, so we use generic version

      LinkedList<Item> lnkList = new LinkedList<Item>();
      LinkListManagement  lx = new LinkListManagement();
      lx.AddInList(lnkList, "Desktop Computer", 3, "A", 0);
      lx.AddInList(lnkList, "Laptop Computer", 8,"A", 0);
      lx.AddInList(lnkList, "Hard Disk", 10, "A", 0);
      lx.AddInList(lnkList, "Pen drive", 25, "A", 0);
      lx.AddInList(lnkList, "Memory Chips", 190, "F", 0);
      lx.AddInList(lnkList, "Keboard", 50, "L", 0);
      lx.AddInList(lnkList, "Mouse", 100, "A", 0);
      lx.AddInList(lnkList, "Light Pen", 20, "I", 1);
      lx.AddInList(lnkList, "Projector", 50, "I", 4);
      lx.AddInList(lnkList, "Projector", 20, "A", 0);

      //  Choose appropriate option to perform operation on linked list
      String opn="";
      do
      {
         // Create a Scanner object for keyboard input.
         Scanner keyboard = new Scanner(System.in);
         // Get the Option
         System.out.println("Enter your Option : ");
         System.out.println("  P (Print the items in LinkedList), D(Sort List by description)");
         System.out.println("  U (Sort List by unit), S (Sort List by description + unit asc order");
         System.out.println("  F (Find an Item [desc.-Projector]), G (Get a sublist index 1-4");
         System.out.println("  M (Edit unit of an Item [desc.- Projector]), R (Replace all desc." +
                                             " Projector by Slid Projector");
         System.out.println("  B (Print List Forward and backword), X (Remove Hard Disk from " +
                                            "List), Y (Remove item at index 3");
         System.out.println("  Z (Remove item in range 0-3), V (Reverse the list), E  (Exit)" );

         opn = keyboard.nextLine();

         switch (opn) {
            case "P" :
               //  Print all Item in the list
               lx.PrintItmList(lnkList);      break;
               //  As our list contains object, it will not print the elements properly
               //  System.out.println("LinkedList contains : " + lnkList);

            case "D" :    //  Sort the list by Description

               //  Sort based on description (ascending). It will use Compare method of
               //  ItemDescComparator comparator defined in Item class
               Collections.sort(lnkList, Item.ItemDescComparator);      //  For Java 8
               //Collections.sort(lnkList, new Item.ItemDescComparator());   //  For Java 7
               System.out.println("Sort on Description ascending order : ");
               lx.PrintItmList(lnkList);     break;

            case "U" :    //  Sort the list by unit

               //  Sort based on unit. It will use CompareTo method defined in Item class
               Collections.sort(lnkList);
               System.out.println("Sort on Unit ascending order : ");
               lx.PrintItmList(lnkList);     break;

            case "S" :    //  Sort the list by description + unit
               //  Sort based on description (ascending) + Units. It will use Compare method of
               //  ItemDescUnitComparator comparator defined in Item class
               Collections.sort(lnkList, Item.ItemDescUnitComparator);       //  For Java 8.0
               //Collections.sort(lnkList, new Item.ItemDescUnitComparator());    //  For Java 7.0

               System.out.println("Sort on Description + Unit ascending order : ");
               lx.PrintItmList(lnkList);     break;

            case "F" :    //  Find an Item by description
               Item itm = new Item("Projector", 0);   //  Find the Projector in the list
```

```
        lx.SearchItm(lnkList, itm);   break;

      case "G" :    //  Get a sublist in the idex 1-4
        lx.GetSubList(lnkList , 1, 4);       break;

      case "M" :    //  Replace the Projector Quantity (first occurance)
        Item newItm = new Item("Projector", 50);
        lx.ReplaceUnit(lnkList,newItm);      break;

      case "R" :    //  Replace all description of an item with a new description
        lx.ReplaceAllDesc(lnkList, "Projector", "Slid Projector");
        break;
      case "B" :    //  Print Link List in forward and backword direction
        lx.PrintAllFrdBck(lnkList);          break;

      case "X" :    //  Remove the specified Item
        Item rdItm = new Item("Hard Disk", 0);
        lx.RemoveItem(lnkList, rdItm, 0, 0);      break;

      case "Y" :    //  Remove Item in index 3
        lx.RemoveItem(lnkList, null, 3, 0);      break;

      case "Z" :    //  Remove range 0-3
        lx.RemoveItem(lnkList, null, 0, 3);      break;

      case "V" :    //  Reverse a Linked list using method of collections framework
        Collections.reverse(lnkList);
        System.out.println("LinkedList is now reversed ....");
        lx.PrintItmList(lnkList);               break;
      default  :
        break;
    }

  } while (opn.charAt(0) != 'E' && opn.charAt(0) != 'e');

}
```