Listing 1: untitled/src/Item.java

```java
abstract class Item implements Comparable<Item> {
    final String name;
    final double price;

    protected Item(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public int compareTo(Item other) {
        int priceCompare = Double.compare(this.price, other.price);
        return priceCompare != 0 ? priceCompare : this.name.compareTo(other.name);
    }

    @Override
    public String toString() {
        return name + ": " + price;
    }
}

class Food extends Item {
    final double weight;

    public Food(String name, double price, double weight) {
        super(name, price);
        this.weight = weight;
    }

    @Override
    public String toString() {
        return super.toString() + " (" + weight + "kg)";
    }
}

class Drink extends Item {
    final double volume;

    public Drink(String name, double price, double volume) {
        super(name, price);
        this.volume = volume;
    }

    @Override
    public String toString() {
        return super.toString() + " (" + volume + "L)";
    }
}
```

Listing 2: untitled/src/ShoppingApp.java

```java
import java.util.Iterator;

public class ShoppingApp {
    public static void main(String[] args) {
        LinkedList<Item> shoppingList = new LinkedList<>();

        shoppingList.add(new Drink("Mineralwasser", 0.79, 0.5));
        shoppingList.add(new Food("Schokolade", 1.0, 0.125));
        shoppingList.add(new Drink("Fanta", 1.39, 1.0));
        shoppingList.add(new Drink("Cola", 1.49, 1.0));
        shoppingList.add(new Food("Brot", 2.39, 0.5));
        shoppingList.add(new Food("Schinken", 17.49, 1.0));

        System.out.println("Shopping␣List␣(Forward):");
        for (Item item : shoppingList) {
            System.out.println(item);
        }

        System.out.println("\nShopping␣List␣(Reverse):");
        Iterator<Item> reverseIterator = shoppingList.reverseIterator();
        while (reverseIterator.hasNext()) {
            System.out.println(reverseIterator.next());
        }

        LinkedList<Item> below2Euro = shoppingList.below(new Food("Any", 2.0, 0));
        System.out.println("\nItems␣below␣2.0␣EUR:");
        for (Item item : below2Euro) {
            System.out.println(item);
        }

        LinkedList<Item> above1Euro = shoppingList.above(new Food("Any", 1.0, 0));
        System.out.println("\nItems␣above␣1.0␣EUR:");
        for (Item item : above1Euro) {
            System.out.println(item);
        }
    }
}
```

Listing 3: untitled/src/LinkedList.java

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class LinkedList<T extends Comparable<? super T>> implements Iterable<T> {
    private static class Node<T> {
        private T data;
        private Node<T> next;
        private Node<T> previous;

        private Node(T data) {
            this.data = data;
            this.next = null;
            this.previous = null;
        }
    }

    private Node<T> head;
    private Node<T> tail;
    private int size = 0;

    public void add(T value) {
        Node<T> newNode = new Node<>(value);
        if (head == null) {
            head = tail = newNode;
        } else {
            Node<T> current = head;
            while (current != null && current.data.compareTo(value) < 0) {
                current = current.next;
            }
```

```java
            if (current == null) {
                tail.next = newNode;
                newNode.previous = tail;
                tail = newNode;
            } else {
                if (current.previous != null) {
                    current.previous.next = newNode;
                    newNode.previous = current.previous;
                } else {
                    head = newNode;
                }
                newNode.next = current;
                current.previous = newNode;
            }
        }
        size++;
    }

    public T get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
        Node<T> current = head;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
        return current.data;
    }

    public T remove(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
        Node<T> current = head;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
        if (current.previous != null) {
            current.previous.next = current.next;
        } else {
            head = current.next;
        }
        if (current.next != null) {
            current.next.previous = current.previous;
        } else {
            tail = current.previous;
        }
        size--;
        return current.data;
    }

    public boolean contains(T value) {
        return indexOf(value) != -1;
    }

    public int indexOf(T value) {
        Node<T> current = head;
        int index = 0;
        while (current != null) {
            if (current.data.equals(value)) {
                return index;
            }
            current = current.next;
            index++;
        }
        return -1;
    }

    public int size() {
        return size;
    }

    @Override
    public Iterator<T> iterator() {
        return new Iterator<T>() {
            private Node<T> current = head;

            @Override
            public boolean hasNext() {
                return current != null;
            }

            @Override
            public T next() {
                if (!hasNext()) throw new NoSuchElementException();
                T data = current.data;
                current = current.next;
                return data;
            }
        };
    }

    public Iterator<T> reverseIterator() {
        return new Iterator<T>() {
            private Node<T> current = tail;

            @Override
            public boolean hasNext() {
                return current != null;
            }

            @Override
            public T next() {
                if (!hasNext()) throw new NoSuchElementException();
                T data = current.data;
                current = current.previous;
                return data;
            }
        };
    }

    public LinkedList<T> below(T value) {
        LinkedList<T> result = new LinkedList<>();
        Node<T> current = head;
        while (current != null && current.data.compareTo(value) <= 0) {
            result.add(current.data);
            current = current.next;
        }
        return result;
    }

    public LinkedList<T> above(T value) {
        LinkedList<T> result = new LinkedList<>();
        Node<T> current = head;
        while (current != null && current.data.compareTo(value) >= 0) {
            result.add(current.data);
            current = current.next;
        }
        return result;
    }




}
```