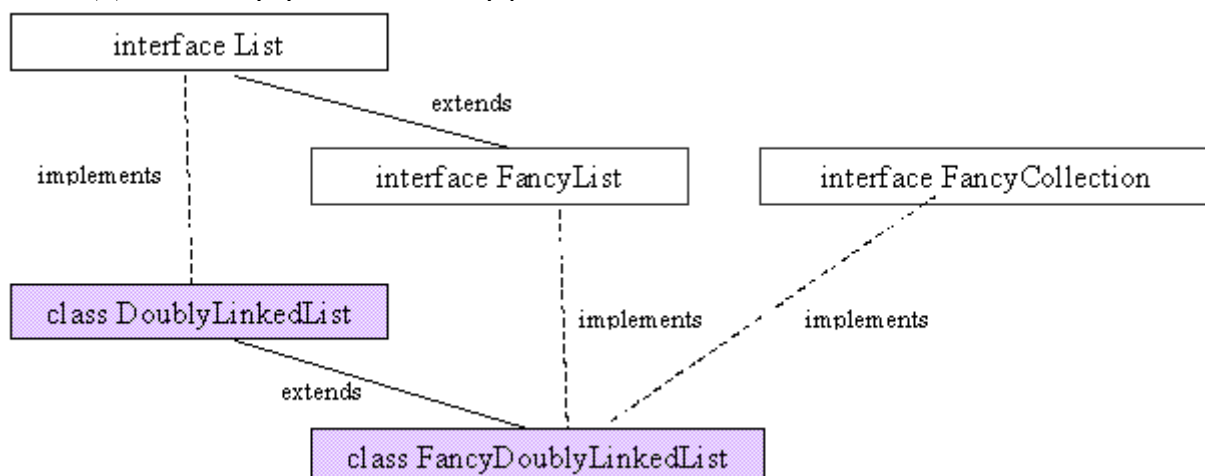


Задача 1: Дадена е йерархията от интерфейси и класове:



Интерфейсът **List** представя операции на списък с елементи от тип **Object**:

```
public interface List {
    public boolean add(Object element);
    public void add(int index, Object element);
    public void addFirst(Object element);
    public void addLast(Object element);
    public boolean contains(Object arg);
    public Object get(int index);
    public Object getFirst();
    public Object getLast();
    public int indexOf(Object element);
    public int lastIndexOf(Object element);
    public boolean isEmpty();
    public Iterator iterator();
    public Object remove(int index);
    public Object remove(Object element);
    public Object removeFirst();
    public Object removeLast();
    public Object set(int index, Object element);
    public int size();
    public String toString();
}
```

Интерфейсът **FancyList** представя операции над списъци:

```
public interface FancyList extends List {
    public boolean containsAll(FancyList c);
    //Returns true if this list contains all of the elements of the specified collection
    public boolean addAll(FancyList c);
    //Appends 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
    //Returns true if this list changed as a result of the call
    //Throws:
    //    UnsupportedOperationException - if the addAll operation is not supported by this list
    public boolean addAll(int index, FancyList c);
    //Inserts all of the elements in the specified collection into this list at the specified position. Shifts the element currently
    //at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will
    //appear in this list in the order that they are returned by the specified collection's iterator
    //Returns true if this list changed as a result of the call
    //Throws:
    //    UnsupportedOperationException - if the addAll operation is not supported by this list
    //    IndexOutOfBoundsException - if the index is out of range (index < 0 || index > size())
}
```

```

        public boolean removeAll(FancyList c);
//Removes from this list all of its elements that are contained in the specified
//collection
//Returns true if this list changed as a result of the call
//Throws:
//    UnsupportedOperationException - if the removeAll operation is not supported by this list
        public boolean retainAll(FancyList c);
//Retains only the elements in this list that are contained in the specified collection. In other words, removes from this
//list all the elements that are not contained in the specified collection
//Returns true if this list changed as a result of the call
//Throws:
//    UnsupportedOperationException - if the retainAll operation is not supported by this list
        public FancyList subList(int fromIndex,int toIndex);
//Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. If
//fromIndex and toIndex are equal, the returned list is empty
//Throws:
//    IndexOutOfBoundsException – if (fromIndex < 0 || toIndex > size() || fromIndex > toIndex)
    }

```

1. Да се реализира **DoublyLinkedList** - линеен двусвързан списък с елементи от тип **Object**, съгласно спецификацията:

```

public class DoublyLinkedList implements List {
    //Представяне
    private DNode head,tail;
    private int num;

    //Методи за достъп
    public DNode getHead() { return head; }
    public void setHead(DNode p) { head = p; }
    public DNode getTail() { return tail; }
    public void setTail(DNode p) { tail = p; }
    public void setNum(int n) { num = n; }
    public int getNum() { return num; }

    //Конструктор, който създава празен списък
    public DoublyLinkedList(){
        setHead(new DNode());
        setTail(new DNode());
        getTail().setLeft(head);
        getHead().setRight(tail);
        num = 0;
    }

    //Собствен метод, който връща указател към възел с индекс index на текущия списък
    private DNode search(int index) throws IndexOutOfBoundsException {
        if(index < 0 || index >= size()) throw new IndexOutOfBoundsException();
        DNode p = getHead().getRight();
        for(int i = 0; i < index; i++) p = p.getRight();
        return p;
    }

    //Реализация на методите на интерфейса List
    //...
    public Iterator iterator() { return new DoublyLinkedListIterator(this); }

    //Друг метод
    public Iterator reverseIterator() { return new DoublyLinkedListReverseIterator(this); }
}

```

Класът **DNode** представя възел на линеен двусвързан списък:

```

public class DNode {
    //Data

```

```

private Object data;
private DNode left,right;

//Constructors
public DNode() { data = left = right = null; }
public DNode(Object d) { data = d; left = right = null; }

//Methods
public DNode getLeft() { return left; }
public void setLeft(DNode l) { left = l; }
public DNode getRight() { return right; }
public void setRight(DNode r) { right = r; }
public Object getData() { return data; }
public void setData(Object d) { data = d; }
}

```

Класът **DoublyLinkedListIterator** представя итератор за обхождане на списък от първия към последния му елемент:

```

public class DoublyLinkedListIterator implements Iterator {
    //Data
    DoublyLinkedList list;
    DNode current;

    //Constructor
    public DoublyLinkedListIterator(DoublyLinkedList list) {
        this.list = list;
        current = list.getHead();
    }

    //Methods
    public boolean hasNext() {return current.getRight() != list.getTail(); }
    public Object next(){
        if(hasNext()){
            current = current.getRight();
            return current.getData();
        }
        throw new java.util.NoSuchElementException();
    }
    public void remove() { throw new UnsupportedOperationException(); }
    public void reset() { current = list.getHead(); }
}

```

Класът **DoublyLinkedListReverseIterator** представя итератор за обхождане на списък от последния към първия му елемент:

```

public class DoublyLinkedListReverseIterator implements Iterator {
    //Data
    DoublyLinkedList list;
    DNode current;

    //Constructor
    public DoublyLinkedListReverseIterator(DoublyLinkedList list) {
        this.list = list;
        current = list.getTail();
    }

    //Methods
    public boolean hasNext() {return current.getLeft() != list.getHead(); }
    public Object next(){
        if(hasNext()){
            current = current.getLeft();
            return current.getData();
        }
    }
}

```

```

        }
        throw new java.util.NoSuchElementException();
    }
    public void remove() { throw new UnsupportedOperationException(); }
    public void reset() { current = list.getTail(); }
}

```

Интерфейсът **Iterator** представя операции на итератор:

```

public interface Iterator {
    public Object next();
    public boolean hasNext();
    public void remove();
    public void reset();
}

```

2. Да се реализира клас **FancyDoublyLinkedList**, съгласно спецификацията:

```

public class FancyDoublyLinkedList extends DoublyLinkedList implements List, FancyList, FancyCollection {
    //Конструктор
    public FancyDoublyLinkedList() {...} //Създава празен списък

    //Наследяване на методите на интерфейса List от клас DoublyLinkedList
    //Реализация на методите на интерфейса FancyList
    // Реализация на методите на интерфейса FancyCollection
}

```