HOMEWORK2

PROBLEM1:

1.1:

```
public static <T> void fndIth(List<T> l, int i) {

		if (!l.empty()) {
			l.findfirst();
			int count = 1;
			while (count != i && (!l.last())) {
				l.findnext();
				count++;
			}
		}
	}
```

1.2:

```
public static <T> void moveBack(List<T> l, int p) {
		int index = 1;
		while (!l.last()) {
			l.findnext();
			index++;
		}

		l.findfirst();
		int size = 1;
		while (!l.last()) {
			l.findnext();
			size++;
		}

		l.findfirst();
		for (int i = 0; i < size - index - p; i++)
			l.findnext();

	}
```

PROBLEM2:

2.1:

```
public void removeK(int k) {
		for (int i = 0; i < k; i++)
			current.next = current.next.next;
	}
```

2.2:

```
	public void insertBeforeCurrent(T e) {
		Node<T> newNode = new Node<T>(e);
		if (current == head) {
			newNode.next = head;
			head = newNode;

		}
		else {
			Node<T> previous = head;

			while (previous.next != current)
				previous = previous.next;

			newNode.next = current;
			previous.next = newNode;
		}
		current = newNode;

	}
```

PROBLEM3:

3.1:

```
	public int direction(T e) {
		if (current.data.equals(e))
			return 0;

		Node<T> temp = head;

		while (temp != current) {
			if (temp.data.equals(e))
				return -1;
			temp = temp.next;
		}

		return 1;

	}
```

3.2:

```
	public static <T extends Comparable<T>> boolean
areReversed(DoubleLinkedList<T> l1, DoubleLinkedList<T> l2) {

		l1.findFirst();
		l2.findFirst();
		int sizeL1 = 1, sizeL2 = 1;
		while (!l1.last()) {
			l1.findNext();
			sizeL1++;
		} // end while

		while (!l2.last()) {
			l2.findNext();
			sizeL2++;
		} // end while

		if (sizeL1 != sizeL2)
			return false;

		l1.findFirst();

		for (int i = 0; i < sizeL1; i++) {

			if (l1.retrieve().compareTo(l2.retrieve()) != 0)
				return false;
			if (!l1.last() && !l2.first()) {
				l1.findNext();
				l2.findPrevious();
			}
		}
		return true;

	}
```

PROBLEM4:

4.1:

```
public static <T> void remove(Queue<T> q, int[] pos, int k) {

		int size = q.length();
		int index = 0;
		T data;

		for (int i = 0; i < size; i++) {
			data = q.serve();
			if (i != pos[index])
				q.enqueue(data);
			else if (index < k - 1)
				index++;

		}

	}
```

4.2:

```
public class ArrayPQ<T> {
	private int maxsize;
	private int size;
	private int head;
	private PQElement<T>[] data;
	// private int[] priority;

	public ArrayPQ(int n) {
		maxsize = n;
		size = 0;
		head = 0;
		data = (PQElement<T>[]) new PQElement<?>[n];
		// priority = new int[n];

	}

	public boolean full() {
		return size == maxsize;
	}

	public int length() {
		return size;
	}

	public PQElement<T> serve() {

		PQElement<T> temp = data[head];
		head++;
		size--;
		return temp;
	}
```

```
	public void enqueue(T e, int pty) {
		PQElement<T> temp = new PQElement<T>(e, pty);

		if ((size == 0))
			data[head] = temp;

		else if (pty > data[head].p)
			if (head != 0) {
				data[--head] = temp;
				size++;
				return;
			}

		if (head != 0) {
			for (int x = 0; x < data.length; x++)
				if (head + x < data.length)
					data[x] = data[head + x];
				else
					data[x] = null;

			head = 0;
		}

		int index = head, i = 0;
		if (!(data[head].p < pty)) {
	for (i = 0; i < size; i++) {
		if (data[index + 1] != null)
			if (data[index].p >= pty && pty > data[index + 1].p) {
					index++;
					break;
				}
				index++;
			}
		}

		for (int j = size - 1; j >= i; j--) {
			data[j + 1] = data[j];
			if (j == index)
				break;
		}

		data[index] = temp;

		size++;
	}
}
```

PROBLEM5:

5.1:

```
public static LinkedList<ItemPair> minPairing(LinkedList<Item> items) {
		LinkedPQ<Item> temp = new LinkedPQ<Item>();
		LinkedList<ItemPair> list = new LinkedList<ItemPair>();
		ItemPair b;
		int size;

		items.findfirst();
		while (!items.last()) {
			temp.enqueue(items.retrieve(), items.retrieve().getPrice());
			items.findnext();
		}
		temp.enqueue(items.retrieve(), items.retrieve().getPrice());

		size = temp.length();

		for (int x = 0; x < size; x = x + 2) {
			b = new ItemPair(temp.serve().data, temp.serve().data);
			list.insert(b);
		}
		if (size % 2 != 0) {
			b = new ItemPair(temp.serve().data, null);
			list.insert(b);
		}

		return list;
	}
```

5.2:

```
public static LinkedList<ItemPair> maxPairing(LinkedList<Item> items) {
		LinkedPQ<Item> temp = new LinkedPQ<Item>();
		LinkedList<ItemPair> list = new LinkedList<ItemPair>();
		ItemPair b;
		int size;

		items.findfirst();
		while (!items.last()) {
			temp.enqueue(items.retrieve(), items.retrieve().getPrice());
			items.findnext();
		}
		temp.enqueue(items.retrieve(), items.retrieve().getPrice());
		size = temp.length();

		LinkedList<Item> tempList = new LinkedList<Item>();
		for (int i = 0; i < size; i++)
			tempList.insert(temp.serve().data);

		Item item1, item2;
```

```
		while (!tempList.empty()) {
			tempList.findfirst();
			item1 = tempList.retrieve();
			tempList.remove();
			if (!tempList.empty()) {
				while (!tempList.last())
					tempList.findnext();

				item2 = tempList.retrieve();
				tempList.remove();
			} else
				item2 = null;

			b = new ItemPair(item1, item2);
			list.insert(b);
		}

		return list;

	}
```

ANTHOER SOLUTION USING ARRAY:

```
public static LinkedList<ItemPair> maxPairing2(LinkedList<Item> items) {
		LinkedPQ<Item> temp = new LinkedPQ<Item>();
		LinkedList<ItemPair> list = new LinkedList<ItemPair>();
		ItemPair b;
		int size;

		items.findfirst();
		while (!items.last()) {
			temp.enqueue(items.retrieve(), items.retrieve().getPrice());
			items.findnext();
		}
		temp.enqueue(items.retrieve(), items.retrieve().getPrice());
		size = temp.length();
		PQElement<Item>[] tempArr = (PQElement<Item>[]) new
PQElement<?>[size];

		for (int i = 0; i < size; i++)
			tempArr[i] = temp.serve();

		for (int x = 0; x < size / 2; x++) {
			b = new ItemPair(tempArr[x].data, tempArr[tempArr.length - (x
+ 1)].data);
			list.insert(b);
		}

		return list;
	}
```

5.3:

When I use the min pairing method:

The First pair would be : 600SR and 400 SR.

The Second pair would be : 200SR and 100 SR.

The Third pair would be: 80SR and 60SR.

The total will be: 1160 SR instead of 1320SR (when I use the max pairing method).

So, I will gain 160SR.