Content: Chapter 4 (Linked Lists), Chapter 5 (Generics, up to, but NOT including interfaces)
Some terms to know: auto-boxing/auto-unboxing, wrapper classes, widening conversion, ADT, generic,
generic type parameter, shallow vs. deep copy (and shallow vs. deep clones),

```java
public class Node {
    private int data;
    private Node link;
    public Node(int initialData, Node initialLink) {
        data = initialData;
        link = initialLink;
    }
    public int getData() { return data; }
    public Node getLink() { return link; }
    public void setData(int newData) { data = newData; }
    public void setLink(Node newLink) { link = newLink; }
}
```

1. Write code (that might be part of an application that uses the Node class) that will create a list
   containing the values (in this order) 5, 31, 23, 4, -3, with 5 being at the head of the list and -3 being
   at the end:

   ```java
   IntNode myList = new IntNode(-3, null);
   myList = new IntNode(4, myList);
   myList = new IntNode(23, myList);
   myList = new IntNode(31, myList);
   myList = new IntNode(5, myList);
   ```

2. Assuming that the linked list in question #1 correctly exists (i.e. `myList` references the head of the
   list), write code (as part of an application that uses the Node class) to print the value of the third
   node in the list.
   ```java
   System.out.print(myList.getLink().getLink().getData());
   ```

3. Write code that creates an Integer wrapper object called `x` and initialize it to 20.
   ```java
   Integer x = new Integer(20);
   ```
Some of the following questions use the `Node<E>` class. Nodes have `E` typed data components. The node's components are accessed through `getLink`, `getData`, `setLink`, and `setData` `Node<E>` class methods.

4. Manipulating linked lists
a. Write a `print` method in Java that takes a reference to `Node<E>` as its only parameter and prints the data in each node on a separate line. It should not change the linked list.

```java
public void print(Node<E> current) {
    while (current != null) {
        System.out.println(current.getData().toString());
        current = current.getLink();
    }
}
```

b. Draw a picture showing what happens in an `addAfter` method that adds the `item` (provided as a parameter) to a new node added after the node given as the parameter `preceding`.

5. Implementing Classes with Linked Lists.
You may assume for this problem that the instance variables of a `Bag` class implemented with a linked list are `head` and `manyNodes`. `head` is a reference to `Node<E>` with nodes having data components of type `E`. `manyNodes` is an integer indicating the number of items in the bag.

a. Why might we use a linked list instead of an array?
Linked lists size automatically as we add nodes or remove nodes. With arrays, we need to resize the array by allocating a new array, then copying all of the items to the new array, which is an expensive operation.

b. Write a `LinkedBag` method `count` which has the method header of

```java
public int count(E target) {
    int c = 0;
    Node<E> current = head;
    while (current != null) {
        if (current.getData().equals(target))
            c++;
        current = current.getLink();
    }
    return c;
}
```

c. Write a `LinkedBag` method that has the method header of

```java
public LinkedBag<E> copyUnique()
```
that returns a new bag that contains the same items as in the current Bag but only one copy of each unique item. No, you haven’t seen this before. Realize that if the Bag you are creating meets its invariant, you may safely call its methods including \texttt{add} and the \texttt{count} method in part.

Do not alter the existing Bag. Do not worry about cloning the individual items, since this is a problem in generic collections.

```java
public LinkedBag<E> copyUnique() {
    LinkedBag<E> newbag = new LinkedBag<E>();
    Node<E> current = head;
    while (current != null) {
        int c = newbag.count(current.getData());
        if (c == 0)
            newbag.add(current.getData());
        current = current.getLink();
    }
    return newbag;
}
```