Main's book Chapter 5
Object Data Types
ArrayBag
Node, Linked Bag

2nd ED Java 1.4
Object types (1)

- So far we have considered ADT's using the primitive `int` type.
- To consider an ADT for another primitive type such as `double` we would need to create a new class for each primitive type.
  - Example: `IntArrayBag` would be rewritten as `DoubleArrayBag`.
- We can use the Object type: one class for all object types.
Object types (2)

- Many ADT's do not use primitive types for the data
- They require data of object type
  - Example: A bag of string objects or a bag of bank account objects
- This does not solve the "generic problem"
  - we would need to write separate bag classes such as `StringBag` or `BankAccountBag`
We have already seen a simple example of a Bank ADT that used BankAccount objects and an array implementation of the Bank collection class.

```java
public class Bank {
    private BankAccount[] bank;
    ...
    ...
    // constructor creates bank;
    ...
```
Object types (4)

- A better approach is to use `Object` type for ADT data. Then we can write a generic class and typecast to any specific type.

- As an example we will convert the `IntArrayBag` class to an Object class called `ArrayBag` that uses an array of type `Object[]` as the data structure.

- Primitive types can be converted to object type using the `wrapper` classes (e.g. `int → Integer`)
The ArrayBag class
Here there are two instance data fields:
(1) `manyItems` gives the number of elements currently in the bag
(2) `data` is a reference to an array of Objects holding the bag elements (the bag elements are references)
After an ArrayBag object is constructed the array of references exists and data is a reference to this array. At any time the first part of the array will hold the bag references and the remaining part of the array is unused.
Example: ArrayBag of strings (1)

```
manyItems 3

data

"Tom" "Dick" "Harry"

used part

unused part
```
Example: ArrayBag of strings (2)

The following statements create an ArrayBag and store some strings in it

```java
ArrayBag bag = new ArrayBag();
bag.add("Tom");
bag.add("Dick");
bag.add("Harry");
```

Now grab some random strings. Type casting is necessary because the return type is Object

```java
String s1 = (String) bag.grab();
String s2 = (String) bag.grab();
```
Example: ArrayBag of strings (3)

This bag has a null reference stored in it (in the used part of the array as opposed to the unused part)
The following statements create an ArrayBag and store some strings in it. This example shows that null references can also be stored in a bag.

```java
ArrayBag bag = new ArrayBag();
bag.add("Tom");
bag.add(null); // add a null reference
bag.add("Harry");
```

Whether or not to allow null references to be stored in a bag is a design decision.

We will allow it here since Main does it this way.
The following statements create an `ArrayBag` and store some integers in it using the `Integer` wrapper class:

```java
ArrayBag bag = new ArrayBag();
bag.add(new Integer(1));
bag.add(new Integer(2));
bag.add(new Integer(3));
```

Now grab some random `Integer` objects and unwrap them. Type casting is necessary because the return type is `Object`:

```java
int i1 = ((Integer) bag.grab()).intValue();
int i2 = ((Integer) bag.grab()).intValue();
```
Java 5 can automatically wrap and unwrap primitive types so we can simplify their use. This is called auto boxing/unboxing.

Example:

```java
ArrayBag bag = new ArrayBag();
bag.add(1); // auto boxing
bag.add(2);
bag.add(3);

Now grab some random Integer objects and unwrap them using auto unboxing

int i1 = (Integer) bag.grab();
int i2 = (Integer) bag.grab();
```
Converting to ArrayBag

- Some important conversion rules
  - Use `==` or `!=` only to test reference equality
  - Use `equals` method to compare distinct objects
  - Decide how a null data item is to be treated
    - allow it in the bag or not: we will allow it
  - Set unused object references to null so garbage collector can remove objects easily
    - otherwise it must wait for bag to be removed
ArrayBag data fields

We now use an array of Object[] type

```java
public class ArrayBag implements Cloneable {

    private Object[] data;
    private int manyItems;
    private static final int INITIAL_CAPACITY = 10;

    ...

```
ArrayBag constructors

Default constructor and one for specified capacity.

```java
public ArrayBag()
{
    this(INITIAL_CAPACITY);
}

public ArrayBag(int initialCapacity)
{
    if (initialCapacity < 0)
        throw new IllegalArgumentException(
            "The initialCapacity is negative: "+ initialCapacity);
    data = new Object[initialCapacity];
    manyItems = 0;
}
```
**add method**

Add a new element to this bag. If the new element would take this bag beyond its current capacity then the capacity is increased before adding the new element.

```java
public void add(Object element) {
    if (manyItems == data.length) {
        // bag is full so expand it
        ensureCapacity(manyItems*2 + 1);
    }
    data[manyItems] = element;
    manyItems++;
}
```

manyItems gives the next available position.
addAll method

Add the contents of another bag to this bag

```java
public void addAll(ArrayBag addend) {
    if (addend == null)
        throw new IllegalArgumentException(  
        "addend is null");

    ensureCapacity(manyItems + addend.manyItems);
    System.arraycopy(addend.data, 0, // all of addend array
                    data, manyItems,// start after end of data
                    addend.manyItems // copy this many items
                      )
    manyItems += addend.manyItems; // update
}
```
Make a clone of this bag.
It's actually a shallow clone since the array is of Object type.
First clone the instance data fields
Then clone the Object array referenced by data

```java
public Object clone()
{
    ArrayBag answer;
    try
    {
        answer = (ArrayBag) super.clone();
    } catch (CloneNotSupportedException e)
    {
        throw new RunTimeException("...");
    }
    answer.data = (Object[]) data.clone();
    return answer;
}
```
public int countOccurrences(Object target) {
    int answer = 0;
    if (target == null) {
        for (int k = 0; k < manyItems; k++) {
            if (data[k] == null) answer++;
        }
    }
    else // continued next slide
}

Process the array from index 0 to index manyItems - 1 and count the number of times the given target object occurs. If the target is null we have to count the null references and we use == for this test.
If the target is not null then we must use the equals method to test for object equality. This assumes that the actual type being referenced by the bag has overridden the equals method.

```java
{ 
    for (int k = 0; k < manyItems; k++)
    {
        if (target.equals(data[k]))
            answer++;
    }
} 
return answer;
```
ensureCapacity method

Change capacity of this bag to at least minimumCapacity. If capacity was already at or greater than minimumCapacity the capacity is left unchanged.

```java
public void ensureCapacity(int minimumCapacity) {
    Object[] biggerArray;
    if (data.length < minimumCapacity) {
        biggerArray = new Object[minimumCapacity];
        System.arraycopy(data, 0, biggerArray, 0, manyItems);
        data = biggerArray;
    }
}
```
getCapacity method

Just return the current maximum size of the data array

```java
public int getCapacity()
{
    return data.length;
}
```
This method wasn't in IntArrayBag (it was in IntLinkedBag).
Generate random index in range 0 to manyItems - 1
Return a reference to object at this position.

```java
public Object grab()
{
    if (manyItems == 0)
        throw new IllegalStateException("...");
    int i = (int)(Math.random()* manyItems);
    return data[i];
}
```
public boolean remove(Object target) {
    int index;
    if (target == null) {
        // use != to find first null
        index = 0;
        while ( (index < manyItems) &&
               (data[index] != null) )
            index++;
    }
    else // see next slide

Search for the given target. If target is not null then we use equals method to search for the first occurrence of the target, remove it, and return true else false.

```java
{  index = 0;
    while ( (index < manyItems) &&
             (! target.equals(data[index])))
        index++;
}
if (index == manyItems) return false;
else
{  manyItems--;
    data[index] = data[manyItems];
    data[manyItems] = null;
    return true;
}
```
Return number of elements currently in the bag

```java
public int size()
{
    return manyItems;
}
```
Reduce that capacity of the bag if necessary to its actual size as given by manyItems. Since manyItems <= data.length the size is reduced only if data.length != manyItems.

```java
public void trimToSize() {
    Object[] trimmedArray;
    if (data.length != manyItems) {
        trimmedArray = new Object[manyItems];
        System.arraycopy(data, 0, trimmedArray, 0, manyItems);
        data = trimmedArray;
    }
}
```
**union method**

Make a new bag which is the union of two given bags

```java
public static ArrayBag union(ArrayBag b1, ArrayBag b2) {
    if (b1 == null)
        throw new IllegalArgumentException("b1 is null");
    if (b2 == null)
        throw new IllegalArgumentException("b2 is null");

    ArrayBag answer = new ArrayBag(
        b1.getCapacity() + b2.getCapacity());
    System.arraycopy(b1.data, 0, answer.data, 0, b1.manyItems);
    System.arraycopy(b2.data, 0, answer.data, b1.manyItems, b2.manyItems);
    answer.manyItems = b1.manyItems + b2.manyItems;
    return answer;
}
```
Main does not do this: String returned has the form ArrayBag[....]. The bag elements are displayed using the objects toString method.

```java
public String toString() {
    StringBuffer s = new StringBuffer();
    s.append("ArrayBag[");
    for (int k = 0; k < size(); k++) {
        s.append(data[k]);
        // don't put comma after last element
        if (k != size() - 1)
            s.append(",");
    }
    s.append("]");
    return s.toString();
}
```
The following statements create an ArrayBag and store some strings in it and then grab one of them (random)

```java
ArrayBag bag = new ArrayBag();
bag.add("Tom");
bag.add("Dick");
bag.add("Harry");
String s = (String) grab();
System.out.println("Bag contains " + bag);
System.out.println("String grabbed is " + s);
```

Typecast is necessary here
There is nothing to stop us from adding objects from any class to our bag. There is no way to prevent a string bag from being used to store other kinds of objects.

```java
ArrayBag bag = new ArrayBag();
bag.add("Tom");
bag.add("Dick");
bag.add("Harry");
bag.add(new Point(1,2));
bag.add(new BankAccount(123,"Fred", 100));
```

To make this illegal generic types were added to Java 5 so that the type of the object that the bag can holds must be specified and it is an error to store other kinds of objects.
The Node Class
for nodes of Object type

2nd ED Java 1.4
Converting IntNode to Node

- We now convert the `IntNode` class of nodes and links to the `Object` type.
- The data type stored in the nodes is now of type `Object` instead of primitive type `int`.
- The new class is called `Node`.
- Note that there are two kinds of null now:
  - a null link that indicates the end of a list
  - a null data item
Node class data fields

Data part of a node is now of type Object

package cs.laurentian.ca.nodes;
public class Node {
    private Object data;
    private Node link;
    // ...

2nd ED Java 1.4
Node class constructor

Data part of a node is now of type Object

```java
public Node(Object initialData, Node initialLink) {
    data = initialData;
    link = initialLink;
}
```
addNodeAfter method

Parameter is now of Object type

```java
public void addNodeAfter(Object element) {
    link = new Node(element, link);
}
```
removeNodeAfter method

No change

```java
public void removeNodeAfter()
{
    link = link.link;
}
```
getData method

Return value is now of Object type

```java
public Object getData()
{
    return data;
}
```
setData method

Parameter is now of Object type

```java
public void setData(Object newData) {
    data = newData;
}
```
Return value is now of Node type

```java
public Node getLink()
{
    return link;
}
```
**setLink** method

*Parameter is now of type Node*

```java
public void setData(Node newLink)
{
    link = newLink;
}
```
**toString method**

Return a string representation of the list whose head is this node. Uses the toString method of the particular object type.

```java
public String toString()
{
    StringBuffer s = new StringBuffer();
    s.append("Node[");
    Node current = this;
    while (current != null)
    {
        s.append(current.data);
        if (current.link != null)
        {
            s.append(",");
            current = current.link;
        }
    }
    s.append("]");
    return s.toString();
}
```
listCopy method (1)

Use the Node type

public static Node listCopy(Node source) {
    if (source == null)
        return null;
}
Use the Node type

```java
Node copyHead = new Node(source.data, null);
Node copyTail = copyHead;

while (source.link != null) {
    source = source.link;
    copyTail.addNodeAfter(source.data);
    copyTail = copyTail.link;
}

return copyHead;
```
Use the Node type

```java
public static Node[] listCopyWithTail(Node source) {
    Node[] answer = new Node[2];
    if (source == null) return answer;
    return answer;
}
```
listCopyWithTail method (2)

Use the Node type

Node copyHead = new Node(source.data, null);
Node copyTail = copyHead;

while (source.link != null)
{
    source = source.link;
    copyTail.addNodeAfter(source.data);
    copyTail = copyTail.link;
}

answer[0] = copyHead;
answer[1] = copyTail;
return answer;
Use the `Node` type

```java
public static int listLength(Node head) {
    int answer = 0;
    for (Node cursor = head; cursor != null; 
        cursor = cursor.link) 
    { 
        answer++;
    }
    return answer;
}
```
listPart method (1)

Use the Node type

```java
public static Node[] listPart(Node start, Node end) {
    if (start == null)
        throw new IllegalArgumentException("...");
    if (end == null)
        throw new IllegalArgumentException("...");
}
```
Use the Node type

```java
Node copyHead = new Node(start.data, null);
Node copyTail = copyHead;
Node cursor = start;
while (cursor != end)
{
    cursor = cursor.link;
    if (cursor == null)
        throw new IllegalArgumentException(".");
    copyTail.addNodeAfter(cursor.data);
    copyTail = copyTail.link;
}
Node[] answer = new Node[2];
answer[0] = copyHead; answer[1] = copyTail;
return answer;
```
Use the Node type

```java
public static Node listPosition(Node head, int position)
{
    if (position <= 0)
        throw new IllegalArgumentException("...");
    Node cursor = head;
    for (int i = 1; (i < position) && (cursor != null); i++)
    {
        cursor = cursor.link;
    }
    return cursor;
}
```
listSearch method (1)

Use the Node type and allow for searching for null data item

```java
public static Node listSearch(Node head, Object target) {
    Node Cursor;

    if (target == null) {
        for(Node cursor = head; cursor != null; cursor = cursor.link)
            if (cursor.data == null)
                return cursor;
    } else // next slide

    else // next slide
```
listSearch method (2)

Use the Node type

```java
{ 
    for (Node cursor = head; cursor != null; 
        cursor = cursor.link) 
        if (target.equals(cursor.data)) 
            return cursor;
    
} 
return null;
}```
The LinkedBag class
(uses the Node class)
Converting IntLinkedListBag

- We can now convert the IntLinkedListBag class to the LinkedListBag class.
- The LinkedListBag class uses the Node class for its implementation so the LinkedListBag class stores data of Object type.
- We can use the static methods in the Node class to implement the LinkedListBag class.
public class LinkedBag implements Cloneable
{
    private Node head;
    private int manyNodes;

    public LinkedBag() {...}
    public void add(Object element) {...}
    public void addAll(LinkedBag addend) {...}
    public Object clone() {...}
    public int countOccurrences(Object target) {...}
    public Object grab() {...}
    public Lister iterator() {...}
    public boolean remove(Object target) {...}
    public int size() {...}
    public static LinkedBag union(LinkedBag b1, LinkedBag b2) {...}
    public String toString() {...}
}
LinkedBag object

LinkedBag object

Linked list of objects in the bag

manyNodes

head

object 0

object 1

object 2
LinkedBag object

object 0

object 1

object 2

10/1/2006

2nd ED Java 1.4
The node type changes from IntNode to the object type Node

public class LinkedBag implements Cloneable {
    private Node head;
    private int manyNodes;
}
LinkedBag constructor

Note: we don't have to worry about capacity so there is just one constructor with no parameters

```java
public LinkedBag()
{
    head = null; // start with empty bag
    manyNodes = 0;
}
```
Since the bag is unordered we can just add new nodes at the head of the list.
The element type is now Object

```java
public add(Object element)
{
    head = new Node(element, head);
    manyNodes++;
}
```
The node type is now Node

```java
public addAll(LinkedBag addend)
{
    if (addend == null)
        throw new IllegalArgumentException("...");
    if (addend.manyNodes > 0)
    {
        Node[] copyInfo = Node.listCopyWithTail(addend.head);
        // link addend copy to this head
        copyInfo[1].setLink(head);
        // set this head to head of copy
        head = copyInfo[0];
        manyNodes += addend.ManyNodes;
    }
}
```
LinkedBag clone method

The node type is now Node

```java
public Object clone()
{
    LinkedBag answer;
    try
    {
        answer = (LinkedBag) super.clone();
    }
    catch (CloneNotSupportedException e)
    {
        throw new RuntimeException("...");
    }
    answer.head = Node.listCopy(head);
    return answer;
}
```
The node type is now Node and parameter is of Object type: 
Note that listSearch was written to allow null data in bag

```java
public int countOccurrences(Object type)
{
    int answer = 0;
    Node cursor = Node.listSearch(head, target);

    while (cursor != null)
    {
        answer++;
        cursor = cursor.getLink();
        cursor = Node.listSearch(cursor,target);
    }
    return answer;
}
```
Here is an alternate method that doesn't use listSearch and is written so it does not allow null data in the bag (equals is used)

```java
public int countOccurrences(Object target) {
    int answer = 0;
    Node cursor = head;
    while (cursor != null) {
        if (target.equals(cursor.getData())) {  
            answer++;
            cursor = cursor.getLink();
        }
    }
    return answer;
}
```
A similar method that uses a standard for loop.

```java
public int countOccurrences(Object target) {
    int answer = 0;
    for (Node cursor = head; current != null; 
        cursor = cursor.getLink())
    {
        if (target.equals(current.getData()))
            answer++;
    }
    return answer;
}
```
LinkedBag grab

Retrieve a random element from this bag.

code:

```java
public Object grab()
{
    if (manyNodes == 0)
        throw new IllegalArgumentException(
            "Bag size is zero");
    int i = (int)(Math.random() * manyNodes) + 1;
    Node cursor = Node.listPosition(head, i);
    return cursor.getData();
}
```
We will explain the Iterator interface later. For now this method just returns an iterator object from a class called Lister.

```java
public Lister iterator()
{
    return new Lister(head);
}
```
The target now has type Object and the node type is Node

```java
public boolean remove(Object target) {
    Node targetNode = Node.listSearch(head, target);
    if (targetNode == null)
        return false;
    else {
        targetNode.setData(head.getData());
        head = head.getLink();
        manyNodes--;
        return true;
    }
}
```
LinkedBag size method

Return number of nodes currently in the list

```java
public int size() {
    return manyNodes;
}
```
public static LinkedBag union(LinkedBag b1, LinkedBag b2) {
    if (b1 == null)
        throw new IllegalArgumentException("...");
    if (b2 == null)
        throw new IllegalArgumentException("...");

    LinkedBag answer = new LinkedBag();
    answer.addAll(b1);
    answer.addAll(b2);
}
public String toString()
{
    StringBuffer s = new StringBuffer();
    s.append("LinkedBag[");
    Node current = head;
    while (current != null)
    {
        s.append(current.getData());
        if (current.getLink() != null)
        {
            s.append(","),
            current = current.getLink();
        }
    }
    s.append("]");
    return s.toString();
}
Iterating over an ADT (1)

To iterate over an ADT means to loop over (or traverse) the ADT one data element at a time and return each data element.

None of our bag classes have provided a way to iterate over the bag data items in this way.

This allows us to sequentially process the data items of an ADT in some order.
Iterating over an ADT (2)

- Internally we have done this using for loops or while loops
  - Example: the toString method uses a while loop to obtain data items one at a time and add them to a string.
  - Example: the listSearch method uses a loop to locate a particular data item.
Iterating over an ADT (3)

We could provide an internal iterator for an ADT in many ways:

Two common ways are

- provide start, advance, isCurrent, and get Current methods in a class such as LinkedBag
- provide methods hasNext and next
Iterating over an ADT (4)

- `public void start();`
  - initialize current element to start of list
- `public boolean isCurrent();`
  - return true if there is a current element
- `public void advance();`
  - advance to the next element in the list
- `public Object getCurrent();`
  - return the current element
Iterating over an ADT (5)

Using **start**, **advance**, **isCurrent**, and **getCurrent** methods in a class such as **LinkedBag** we would use the statements:

```java
LinkedBag bag = new LinkedBag();
// add some items to the bag here
for (bag.start(); bag.isCurrent(); bag.advance())
{
    // use bag.getCurrent() to retrieve the
    // current element and do something with it
}
```
Iterating over an ADT (6)

- **public void start();**
  - initialize current element to start of list

- **public boolean hasNext();**
  - return true if there is a next element

- **public Object next();**
  - return the current element and advance to the next one.
Using `start`, `hasNext` and `next` methods in a class such as `LinkedBag` we would use the statements

```java
LinkedBag bag = new LinkedBag();
// add some items to the bag here

bag.start();
while (bag.hasNext())
{
    // use bag.next() to get data item
    // and advance to next one.
}
```
A more useful approach is to implement the iterator as an external class that implements the standard `Iterator` interface.

```java
public Interface Iterator {
    public boolean hasNext();
    public Object next();
    public void remove();
}
```
The Lister class iterator

- We will write a Lister class that implements the `Iterator` interface.
- This class can be used by `LinkedBag` to return an instance of `Lister` using its iterator method:

```java
public Lister iterator()
{
    return new Lister(head);
}
```
While loop Iterator

```java
LinkedBag bag = new LinkedBag();

bag.add(new BankAccount(123, "Jack", 100.0));
bag.add(new BankAccount(124, "Jill", 150.0));
...

Iterator iter = bag.iterator(); // get iterator

while (iter.hasNext())
{
    BankAccount b = (BankAccount) iter.next();
    // do something here with bag element
}
```
For loop Iterator

```java
LinkedListBag bag = new LinkedListBag();

bag.add(new BankAccount(123, "Jack", 100.0));
bag.add(new BankAccount(124, "Jill", 150.0));
...

BankAccount b;
for( Iterator iter = bag.iterator();
    iter.hasNext();
    b = (BankAccount) iter.next() )
{
    // do something with b here
}
```
Unified modelling language

Part of UML includes a notation for expressing relationships among classes
- inheritance "is a"
- implements "is a"
- composition "is a part of", "has a"
- association "uses"

BlueJ uses a simple form of UML diagrams for classes
UML Diagram notation

Class Diagram

<table>
<thead>
<tr>
<th>ClassName</th>
<th>Instance variables</th>
<th>methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;abstract&gt;&gt;</td>
<td>Instance variables</td>
<td>methods</td>
</tr>
<tr>
<td>InterfaceName</td>
<td>&lt;&lt;interface&gt;&gt;</td>
<td>methods</td>
</tr>
</tbody>
</table>

Inheritance ("is a")

Implements ("is a")

Composition ("is a part of")

Association ("uses")

10/1/2006

BGA
**UML class diagram**

**BankAccount**
- number
- name
- balance
- + deposit
- + withdraw
- + getNumber
- + getName
- + setName
- + getBalance
- + compareTo
- + equals
- + clone
- + toString

**JointBankAccount**
- jointName
- + getJointName
- + setJointName
- + getBalance
- + equals
- + clone
- + toString

**Bank**
- + size
- + add
- + get
- + set
- + clone
- + toString

**Comparable**
- <<interface>>
- + compareTo
Compact UML class diagram

- `BankAccount`
- `JointBankAccount`
- `Bank`

Relationships:
- 1..m "is a part of" relationship between `BankAccount` and `Bank`.
- "is a" relationship between `JointBankAccount` and `BankAccount`.

Interfaces:
- `Comparable` <<interface>>
- `Serializable` <<interface>>
- `Cloneable` <<interface>>
LinkedBag class diagram

- Node
- LinkedBag
- Lister
- <<interface>> Iterator

This class just needs to implement the `Iterator` interface.

```java
public class Lister implements Iterator {
    private Node list;

    public Lister(Node head) {
        list = head;
    }

    // continued next slide

give this constructor a reference to the head of the list you want to iterate over
```
This class just needs to implement the Iterator interface.

```java
public boolean hasNext()
{
    return list != null;
}
public Object next()
{
    if (!hasNext())
        throw new NoSuchElementException("...");
    Object answer = list.getData();
    list = list.getLink(); // next entry
    return answer;
}
```

// next slide
We do not want to implement the remove method so we just throw an exception to indicate that it is not implemented.

```java
public void remove()
{
    throw new UnsupportedOperationException(  
        "Remove not implemented"  
    );
}
```
Create a `LinkedBag` of strings, grab a few strings and display them.

```java
LinkedBag bag = new LinkedBag();

bag.add("Tom"); bag.add("Dick");
bag.add("Harry");

String s1 = (String) bag.grab();
String s2 = (String) bag.grab();

System.out.println("Strings grabbed are " + s1 + " and " + s2);
```
Create a `LinkedBag` of `Integer` objects, grab a few `Integers` and display them.

```java
LinkedList bag = new LinkedList();
bag.add(new Integer(1));
bag.add(new Integer(2));
bag.add(new Integer(3));

int i1 = ((Integer) bag.grab()).intValue();
int i2 = ((Integer) bag.grab()).intValue();

System.out.println("Integers grabbed are " + i1 + " and " + i2);
```
BankAccount bag example (1)

Create a LinkedList of BankAccount objects and use the bag iterator to display them

```java
LinkedList bag = new LinkedList();

bag.add(new BankAccount(123, "Jack", 100.0));
bag.add(new BankAccount(124, "Jill", 150.0));
bag.add(null); bag.add(null);

Lister iter = bag.iterator();
while (iter.hasNext())
{
    BankAccount b = (BankAccount) iter.next();
    System.out.println(b);
}
```
Create a bag of BankAccount objects and display only the ones whose balance is less than 100 dollars

```java
LinkedBag bag = new LinkedBag();
bag.add(new BankAccount(123, "Jack", 100.0));
bag.add(new BankAccount(124, "Jill", 150.0));
// add more accounts here

Lister iter = bag.iterator();
while (iter.hasNext())
{
    BankAccount b = (BankAccount) iter.next();
    if (b.getBalance() < 100.0)
    {
        System.out.println(b);
    }
}
```
Write an ArrayLister class that can be used with ArrayBag. Put the following method in the ArrayBag class and then write the ArrayLister class that uses the given current position.

```java
public ArrayLister iterator()
{
    return new ArrayLister(data, manyItems);
}
```

Here the arguments are the data array holding the bag elements and manyItems to indicate how many elements are currently in the bag.
Use box like this for comment

Put code here