# Object-Oriented Principles in Java: Part II

Issues associated with objects containing/composed of other objects:

- •Composition, Reuse

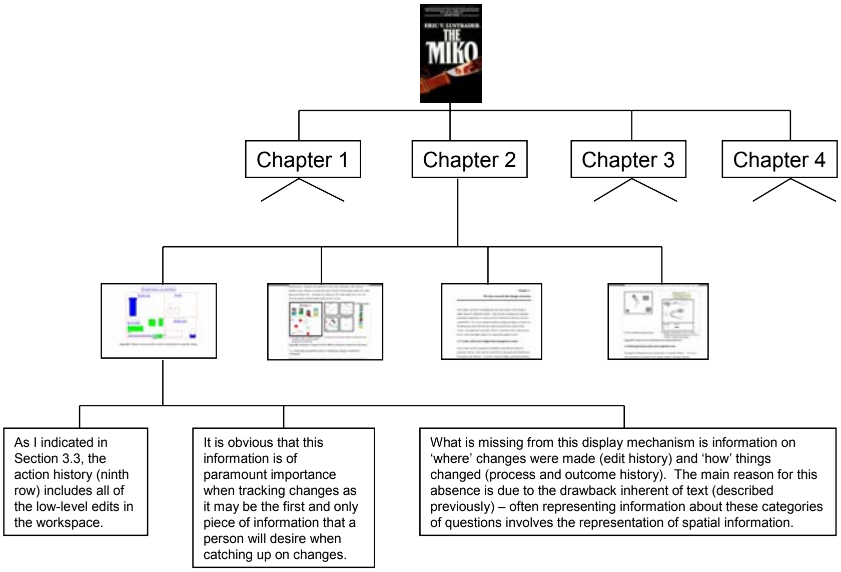
Issues associated with object references

- •Assignment, comparisons

Useful for operations

- •The String class/Displaying object attributes

# Composition: Books



Chapter 1

Chapter 2

Chapter 3

Chapter 4

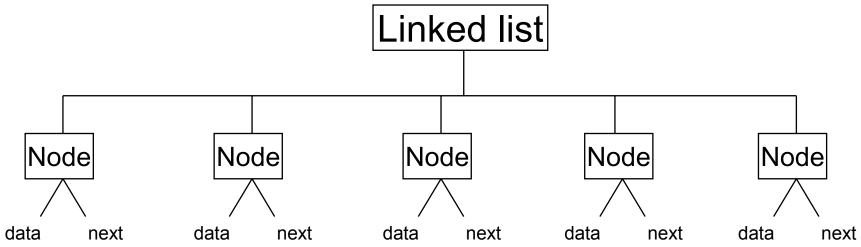
As I indicated in Section 3.3, the action history (ninth row) includes all of the low-level edits in the workspace.

It is obvious that this information is of paramount importance when tracking changes as it may be the first and only piece of information that a person will desire when catching up on changes.

What is missing from this display mechanism is information on ‘where’ changes were made (edit history) and ‘how’ things changed (process and outcome history). The main reason for this absence is due to the drawback inherent of text (described previously) – often representing information about these categories of questions involves the representation of spatial information.

# Composition: Lists

Assume that a list has been implemented as a linked list.
The list can be again viewed as a hierarchy:



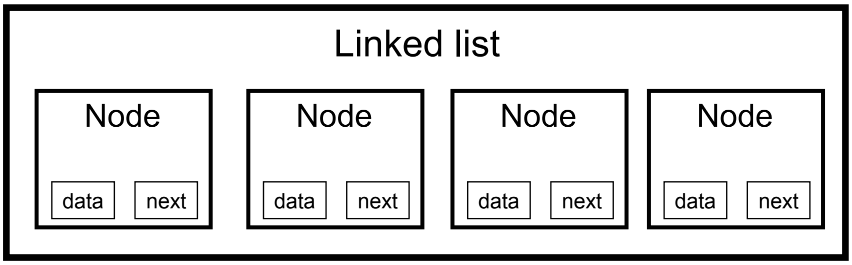
Linked list

Node Node Node Node Node

data next data next data next data next data next

# Composition: Lists

Alternative representation (containment):

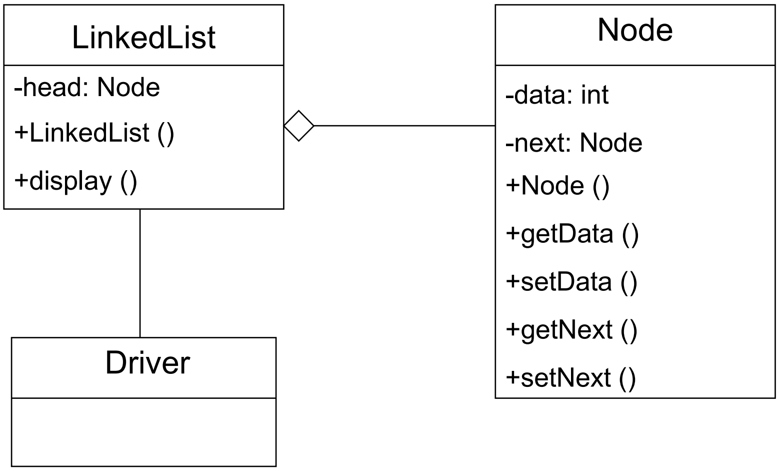


Linked list

Node Node Node Node

data next data next data next data next

# A Simple Linked List Implemented In Java



LinkedList

-head: Node

+LinkedList ()

+display ()

Node

-data: int

-next: Node

+Node ()

+getData ()

+setData ()

+getNext ()

+setNext ()

Driver

# Composition: The Driver Class

The following example can be found in the directory:
/home/profs/tamj/233/examples/composition

```
class Driver
{
   public static void main (String [] argv)
   {
      LinkedList integerList = new LinkedList ();
      integerList.display();
   }
}
```

# Composition: The Linked List Class

```
class LinkedList
{
   private Node head;

   public LinkedList ()
   {
       int i = 1;
      Node temp;
      head = null;
      for (i = 1; i <= 4; i++)
      {
         temp = new Node ();
         temp.setNext(head);
         head = temp;
      }
   }
```

# Composition: The Linked List Class (2)

```
   public void display ()
   {
      int i = 1;
      Node temp = head;
      while (temp != null)
      {
         System.out.println("Element No. " + i + "=" + temp.getData());
         temp = temp.getNext();
         i++;
      }
   }
}
// End of class Linked List
```

# Composition: The Node Class

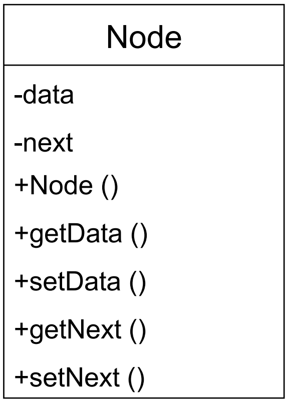
```
import java.util.Random;

class Node
{
   private int data;
   private Node next;

   Node ()
   {
      data = (int) (Math.random() * 100);
      next = null;
   }

   public int getData ()
  {
      return data;
  }
```

# Composition: The Node Class (2)

```
   public void setData (int num)
   {
      data = num;
   }

   public Node getNext ()
   {
      return next;
   }

   public void setNext (Node nextNode)
   {
      next = nextNode;
   }
}

// End of class Node
```

# Composition And Code Reuse

```
Class Linked List
{
    Node temp = new Node ();
                 :
}
```



Node

-data

-next

+Node ()

+getData ()

+setData ()

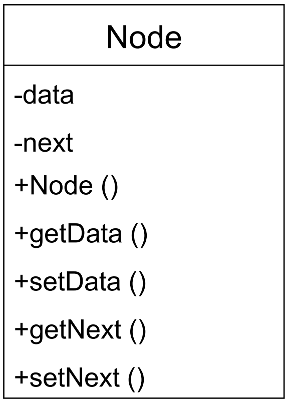
+getNext ()

+setNext ()

# Composition And Code Reuse

```
Class Linked List
{
    Node temp = new Node ();
                 :
}
```

It's "for free"



Node

-data

-next

+Node ()

+getData ()

+setData ()

+getNext ()

+setNext ()

# Composition: Alternative Names

“Whole-part”
“Has-A”
“Includes” / “Part-of”

# Issues Associated With Object References

Assignment
Comparisons

# Issues Associated With Object References

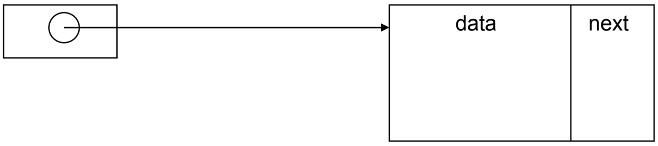
**Assignment**

Comparisons

# Reference: Can’t Be De-referenced By Programmer

Node temp;

temp = new Node ();



# Assignment Operator: Works On The Reference

```
Node n1 = new Node ();
Node n2 = new Node ();
n2 = n1;
n2.setData(888);
```

# Copying Data Between References

**Perform a field-by-field copy**

Clone the object

# Field-By-Field Copy

```
 class IntegerWrapper
{
   private int num;
   public void setNum (int no)
   {
      num = no;
   }

   public int getNum ()
   {
      return num;
   }
```

# Comparisons: Comparing The References

```
Node n1 = new Node ();
Node n2 = new Node ();
if (n1 == n2)
   System.out.println("Same node");
else
  System.out.println("Two different nodes");

n2 = n1;
if (n1 == n2)
  System.out.println("Same node");
else
  System.out.println("Two different nodes");
```

# Comparing Data For References

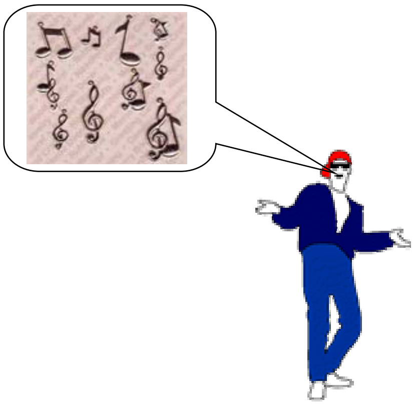
Use equals ()

```
Node n1 = new Node ();
Node n2 = new Node ();

if (n1.equals(n2))
    System.out.println("Equal data");
else
    System.out.println("Data not equal");
```

# Passing By Reference For Simple Types

It can be done in Java

Just use a wrapper!



# Passing By Reference For Simple Types (2)

```
class IntegerWrapper
{
   private int num;
   public int getNum () { return num; }
   public void setNum (int no) { num = no; }
}

class Driver
{
   public static void method (IntegerWrapper temp) { temp.setNum(10); }

    public static void main (String [] argv)
    {
       IntegerWrapper temp = new IntegerWrapper ();
       temp.setNum(1);
       method(temp);
    }
}
```

# The String Class Revisited

A Java class but the attribute fields can be displayed directly (via print/println)

A String is created like a simple type (when double quotes are encountered).

Any of the simple types will be converted to a string when passed to the print/println method.

String concatenation:

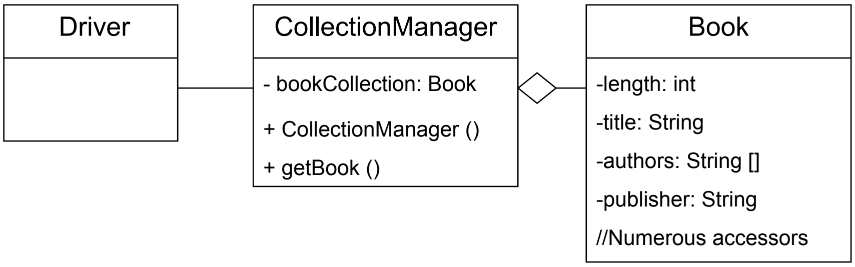
•Plus sign: e.g., System.out.println("Str1" + "Str2");

•Method: e.g., public String concat (String str)

For more detailed information regarding the String class goto the url:

http://java.sun.com/j2se/1.4/docs/api/java/lang/String.html

# A Simple Book Collection Example



The full example can be found in the directory:

/home/profs/tamj/233/examples/displayingClasses/fieldByField

# The Driver Class

```
import tio.*;

class Driver
{
   public static void main (String [] argv)
   {
      int i, j;
      Book tempBook;
      CollectionManager tamjCollection = new CollectionManager ();
      System.out.println("\nJAMES' BOOK COLLECTION");
      for (i = 0; i < 80;  i++)
      System.out.print("-");
      System.out.println();
```

# The Driver Class (2)

```
    for (i = 0; i < CollectionManager.NOBOOKS; i++)
    {
        System.out.println("\tBook: " + (i+1));
        tempBook = tamjCollection.getBook(i);
        System.out.println("\tTitle..." + tempBook.getTitle());
        System.out.println("\tLength..." + tempBook.getLength() + " pages");
        System.out.print("\tAuthors:   ");
        for (j = 0; j < CollectionManager.NOAUTHORS; j++)
              System.out.print(tempBook.getAuthorAt(j) + "  ");
        System.out.println();
        System.out.println("\tPublisher..." + tempBook.getPublisher());
        for (j = 0; j < 80; j++)
        System.out.print("~");
        System.out.println("Hit return to continue");
        Console.in.readChar();
    }
  }
}
```

# The Collection Manager Class

```
class CollectionManager
{
   private Book [] bookCollection;
   public static final int NOBOOKS = 4;
   public static final int NOAUTHORS = 3;

   public CollectionManager ()
   {
      int i;
      bookCollection = new Book[NOBOOKS];
      for (i = 0; i < NOBOOKS; i++)
      {
            bookCollection[i] = new Book ();
      }
   }
```

# The Collection Manager Class (2)

```
   public Book getBook (int index)
   {
      return bookCollection[index];
   }
}
```

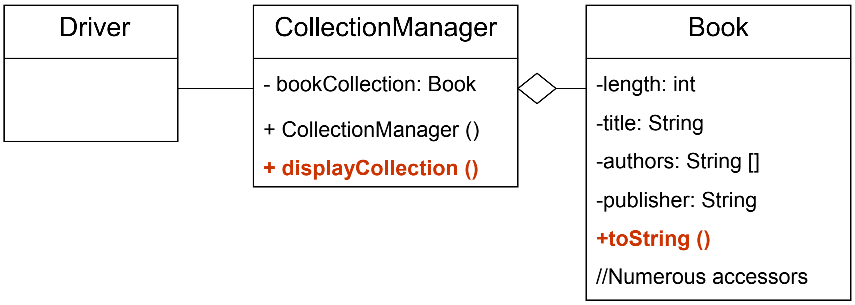
# Portions Of The Book Class

```
class Book
{
   private int length;
   private String title;
   private String [] authors;
   private String publisher;
   private static int seriesNumber = 1;
   private static final int NOAUTHORS = 3;
```

# Portions Of The Book Class (2)

```
    public Book ()
    {
       int i;
       length = (int) (Math.random() * 900) + 100;
       title = "How to series, Number " + Book.seriesNumber;
       Book.seriesNumber++;
       authors = new String[NOAUTHORS];
       for (i = 0; i < NOAUTHORS; i++)
       {
          authors[i] = "Author-" + (i+1);
       }
       publisher = "Book publisher";
    }

   // Numerous accessor (get/set) methods.
   // A second constructor
}
```

# A Revised Version Of The Book Example



Driver

CollectionManager
- bookCollection: Book
+ CollectionManager ()
**+ displayCollection ()**

Book
-length: int
-title: String
-authors: String []
-publisher: String
**+toString ()**
//Numerous accessors

The full example can be found in the directory:

/home/profs/tamj/233/examples/displayingClasses/toString

# The Driver Class

```
class Driver
{
   public static void main (String [] argv)
   {
      int i;
      Book tempBook;
      CollectionManager tamjCollection = new CollectionManager ();
      System.out.println("\nJAMES' BOOK COLLECTION");
      for (i = 0; i < 80; i++)
         System.out.print("-");
      System.out.println();
      tamjCollection.displayCollection();
   }
}
```

# The CollectionManager Class

```
import tio.*;

class CollectionManager
{
   private Book [] bookCollection;
   public static final int NOBOOKS = 4;

   public CollectionManager ()
   {
      int i;
      bookCollection = new Book[NOBOOKS];
      for (i = 0; i < NOBOOKS; i++)
      {
            bookCollection[i] = new Book ();
      }
   }
```

# The Collection Manager Class (2)

```
    public void displayCollection ()
    {
        int i;
        for (i = 0; i < NOBOOKS; i++)
        {
            System.out.println(bookCollection[i]);
            System.out.println("Hit return to continue");
            Console.in.readChar();
        }
    }
}
```

# The Book Class

```
class Book
{
    private int length;
    private String title;
    private String [] authors;
    private String publisher;
    private static final int NOAUTHORS = 3;
    private static int seriesNumber = 1;
```

# The Book Class (2)

```
public String toString ()
{
   String allFields = new String ();
   int i;
   allFields = allFields.concat("\tBook Title..." + title + "\n");
   allFields = allFields.concat("\tLength..." + length + " pages\n");
   allFields = allFields.concat("\tAuthors:   ");
   for (i = 0; i < authors.length; i++)
   {
      allFields = allFields.concat(authors[i] + "  ");
   }
   allFields = allFields.concat("\n");
   allFields = allFields.concat("\t" + publisher + "\n");
   for (i = 0; i < 80; i++)
      allFields = allFields.concat("~");
   allFields = allFields.concat("\n");
   return allFields;
}
```

# The Book Class (2)

Automatically called when an instance of the class is passed as parameter to print/println

```
public String toString ()
{
   String allFields = new String ();
   int i;
   allFields = allFields.concat("\tBook Title..." + title + "\n");
   allFields = allFields.concat("\tLength..." + length + " pages\n");
   allFields = allFields.concat("\tAuthors:   ");
   for (i = 0; i < authors.length; i++)
   {
      allFields = allFields.concat(authors[i] + "  ");
   }
   allFields = allFields.concat("\n");
   allFields = allFields.concat("\t" + publisher + "\n");
   for (i = 0; i < 80; i++)
      allFields = allFields.concat("~");
   allFields = allFields.concat("\n");
   return allFields;
}
```

# Summary

You should now know:

- What composition means in terms of Object-Oriented theory and how to implement it in Java.
- How assignment and comparisons work with Java objects (references)
- Another example of why implementation hiding is a useful principle with the creation of the toString () method.