Java Arrays (review)
Linked Lists (preview)

Array Agenda

- What is an array
- Declaration of an array
- Instantiation of an array
- Accessing array element
- Array length
- Multi-dimensional array

What is an Array?

Introduction to Arrays

- Suppose we have here three variables of type int with different identifiers for each variable.

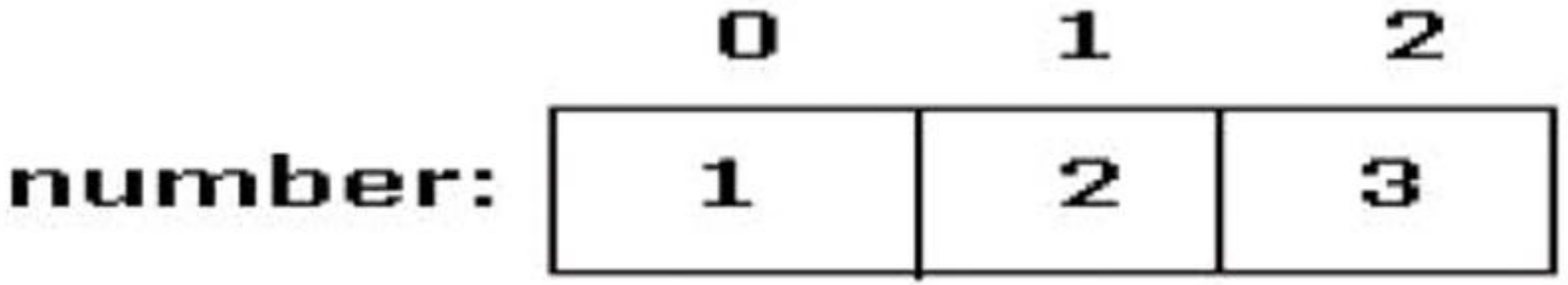
```
int number1;
int number2;
int number3;

number1 = 1;
number2 = 2;
number3 = 3;
```

As you can see, it seems like a tedious task in order to just initialize and use the variables especially if they are used for the same purpose.

Introduction to Arrays

- In Java and other programming languages, there is one capability wherein we can use one variable to store a list of data and manipulate them more efficiently. This type of variable is called an array.

- An array stores multiple data items of the same data type, in a contiguous block of memory, divided into a number of slots.



Declaration of an Array

Declaring Arrays

- To declare an array, write the data type, followed by a set of square brackets[], followed by the identifier name.

- For example,

```
int []ages;
```

or

```
int ages[];
```

Instantiation of an Array

Array Instantiation

- After declaring, we must create the array and specify its length with a constructor statement.

- Definitions:
  - Instantiation
    - In Java, this means creation
  - Constructor
    - In order to instantiate an object, we need to use a constructor for this. A constructor is a method that is called to create a certain object.
    - We will cover more about instantiating objects and constructors later.

Array Instantiation

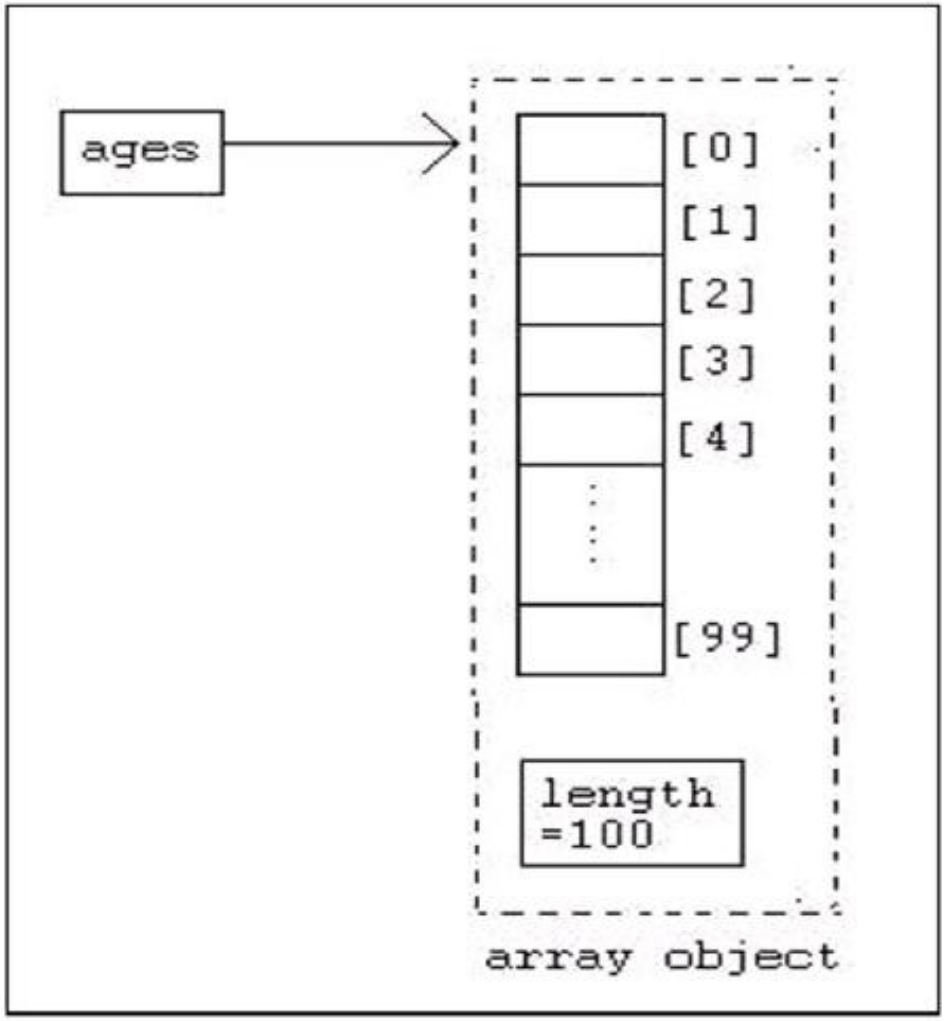
- To instantiate (or create) an array, write the `new` keyword, followed by the square brackets containing the number of elements you want the array to have.
- For example,

```
//declaration
int ages[];

//instantiate object
ages = new int[100];
```

or, can also be written as,

```
//declare and instantiate object
int ages[] = new int[100];
```

Array Instantiation



Array Instantiation

- You can also instantiate an array by directly initializing it with data.

- For example,

```
int arr[] = {1, 2, 3, 4, 5};
```

  This statement declares and instantiates an array of integers with five elements (initialized to the values 1, 2, 3, 4, and 5).

Sample Program

```
1   //creates an array of boolean variables with identifier
2   //results. This array contains 4 elements that are
3   //initialized to values {true, false, true, false}
4
5   boolean results[] = { true, false, true, false };
6
7   //creates an array of 4 double variables initialized
8   //to the values  {100, 90, 80, 75};
9
10  double []grades = {100, 90, 80, 75};
11
12  //creates an array of Strings with identifier days and
13  //initialized. This array contains 7 elements
14
15  String days[] = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
    "Sun"};
```

Accessing Array Element

Accessing an Array Element

- To access an array element, or a part of the array, you use a number called an index or a subscript .

- index number or subscript
  - assigned to each member of the array, to allow the program to access an individual member of the array.
  - begins with zero and progress sequentially by whole numbers to the end of the array.
  - NOTE: Elements inside your array are from 0 to (sizeOfArray-1).

Accessing an Array Element

- For example, given the array we declared a while ago, we have

```
//assigns 10 to the first element in the array
ages[0] = 10;

//prints the last element in the array
System.out.print(ages[99]);
```

Accessing an Array Element

- NOTE:
  - once an array is declared and constructed, the stored value of each member of the array will be initialized to zero for number data.
  - for reference data types such as Strings, they are NOT initialized to blanks or an empty string “”. Therefore, you must populate the String arrays explicitly.

Accessing an Array Element

- The following is a sample code on how to print all the elements in the array. This uses a for loop, so our code is shorter.

```
1  public class ArraySample{
2     public static void main( String[] args ){
3        int[] ages = new int[100];
4        for( int i=0; i<100; i++ ){
5           System.out.print( ages[i] );
6        }
7     }
8  }
```

Coding Guidelines

1. It is usually better to initialize or instantiate the array right away after you declare it. For example, the declaration,

```
int []arr = new int[100];
```

   is preferred over,

```
int []arr;
arr = new int[100];
```

Coding Guidelines

2. The elements of an n-element array have indexes from 0 to n-1. Note that there is no array element arr[n]! This will result in an array-index-out-of-bounds exception.

3. Remember: You cannot resize an array.

Array Length

Array Length

- In order to get the number of elements in an array, you can use the length field of an array.

- The length field of an array returns the size of the array. It can be used by writing,

```
arrayName.length
```

Array Length

```
1  public class ArraySample {
2     public static void main( String[] args ){
3        int[] ages = new int[100];
4
5        for( int i=0; i<ages.length; i++ ){
6           System.out.print( ages[i] );
7        }
8     }
9  }
```

Coding Guidelines

1. When creating for loops to process the elements of an array, use the array object's length field in the condition statement of the for loop. This will allow the loop to adjust automatically for different-sized arrays.

2. Declare the sizes of arrays in a Java program using named constants to make them easy to change. For example,

```
final int ARRAY_SIZE = 1000; //declare a constant

int[] ages = new int[ARRAY_SIZE];
```

Multi-Dimensional Array

Multidimensional Arrays

- Multidimensional arrays are implemented as arrays of arrays.
- Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name.

Multidimensional Arrays

- For example,

```
// integer array 512 x 128 elements
int[][] twoD = new int[512][128];

// character array 8 x 16 x 24
char[][][] threeD = new char[8][16][24];

// String array 4 rows x 2 columns
String[][] dogs = {{ "terry", "brown" },
                   { "Kristin", "white" },
                   { "toby", "gray"},
                  { "fido", "black"}
                  };
```

Multidimensional Arrays

- To access an element in a multidimensional array is just the same as accessing the elements in a one dimensional array.

- For example, to access the first element in the first row of the array dogs, we write,

```
System.out.print( dogs[0][0] );
```

  This will print the String "terry" on the screen.

Summary

- Arrays
  - Definition
  - Declaration
  - Instantiation and constructors (brief overview - to be discussed more later)
  - Accessing an element
  - The length field
  - Multidimensional Arrays

Linked Lists Agenda

- Learn about *linked lists*.
- Get used to thinking about more than one possible implementation of a data structure.
- Think about the advantages and disadvantages of different implementations.

Reading

- Carrano Chapter 3

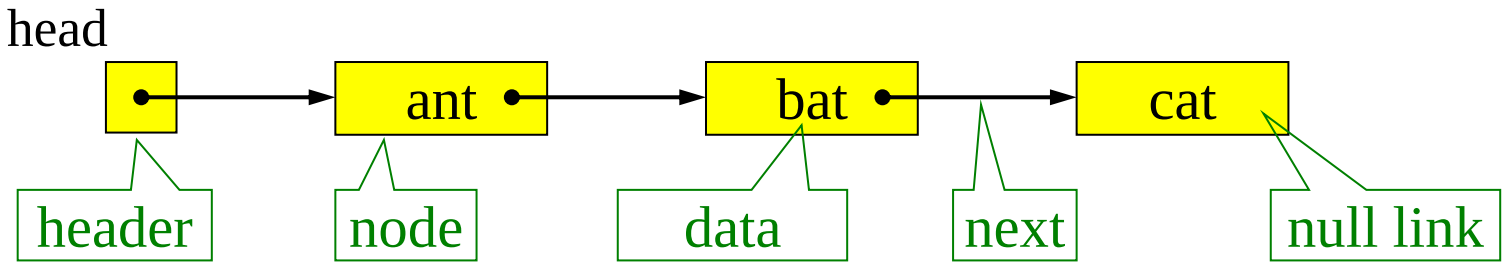
Linked lists: the idea

- A *linked list* is a set of items where each item is part of a *node* that may also contain a single *link* to another node.

- Allow one to insert, remove and rearrange lists very efficiently.

Linked lists: data structure

- A **linked list** consists of a sequence of **nodes** connected by **links**, plus a **header**.
- Each node (except the last) has a **next** node, and each node (except the first) has a **predecessor**.
- Each node contains a single **data** element (object or value), plus links to the **next** Node..



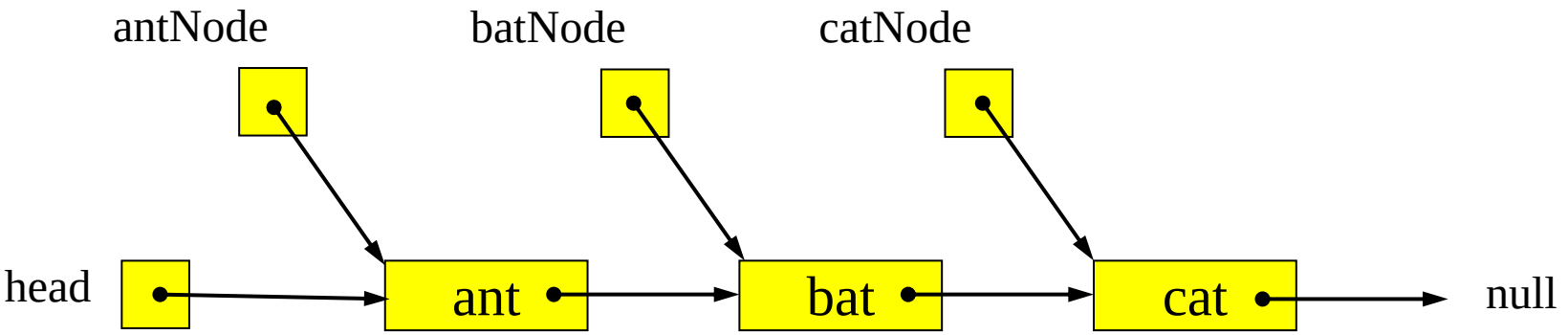
A Java Class of Nodes

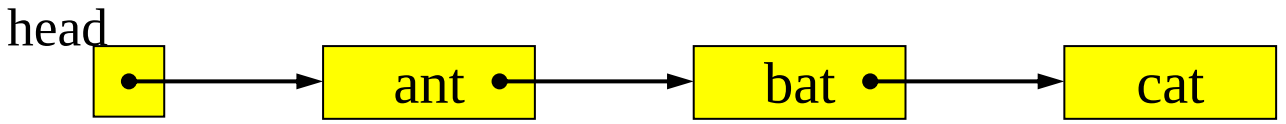
- Nodes for a linked list (of Objects).

```
class Node{
    Object data;                          // data field
    Node next;                            // next field

    // Constructor
    Node (Object data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```

Example of list creation

```
Node catNode = new Node("cat",null);
Node batNode = new Node("bat",catNode);
Node antNode = new Node("ant",batNode);
Node head = antNode;
```



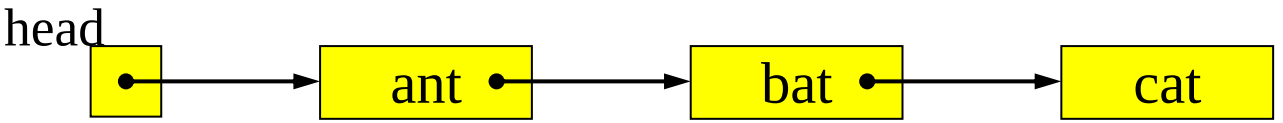


What happens?

- System.out.println(head.data);
- System.out.println(head.next.data);
- System.out.println(head.next.next.data);
- System.out.println(head.next.next.next.data);

More about linked lists

- The **length** of a linked list is the number of nodes.
- An **empty** linked list has no nodes.
- In a linked list:
  - We can *manipulate* the individual *elements*.
  - We can *manipulate* the *links*,
    - Thus we can change the structure of the linked list!
    - This is not possible in an array.

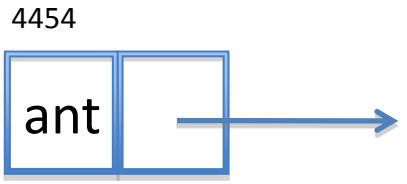


// printing with a loop -- List Traversal

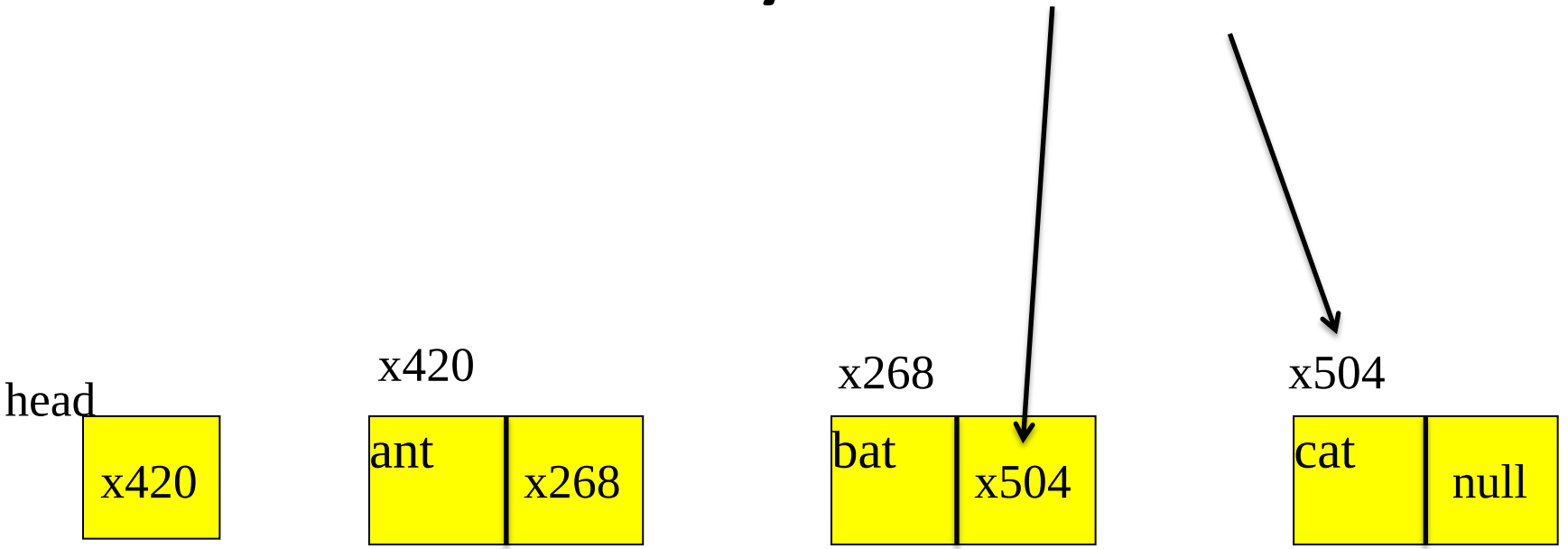
```
Node p = head;
while(p!=null)
{
   System.out.print(p.data + "->");
   p = p.next;
}
System.out.println("null");

// printing with printList method
printList(head);
```

Points to Note

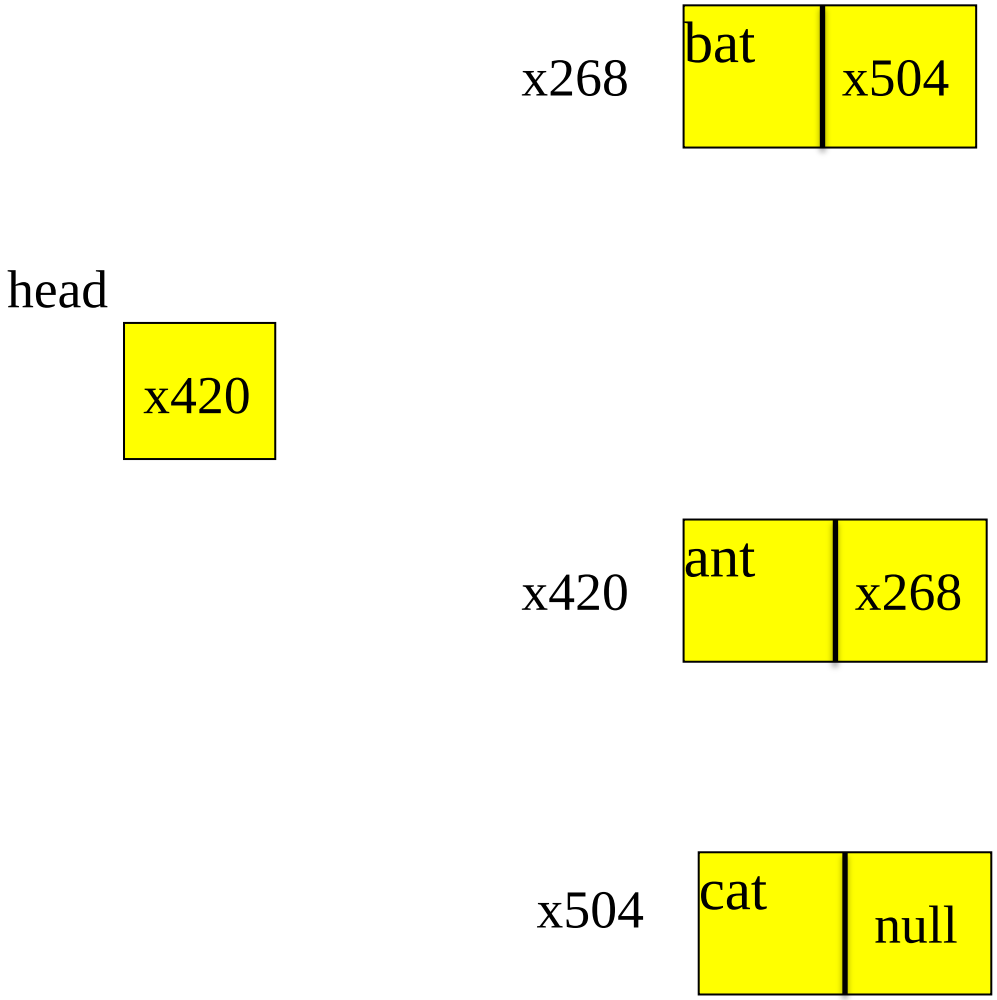
- Last element uses a special link called the ***null link***.
- Different implementations of linked lists.
- Different forms:
  - **Linear** lists: items in sequential order
  - **Circular** lists: `last' item linked to `first'.
  - **Cyclic**: `last' item linked to one of its predecessors.
  - Nodes sometimes drawn:

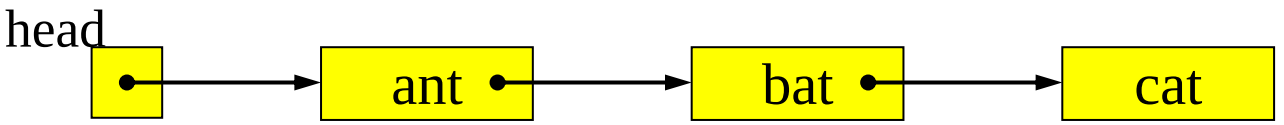


Memory addresses



Memory addresses





//******* demo 2 Insert a Node "asp" at front**

// 1) create a new node with desired data ("asp")

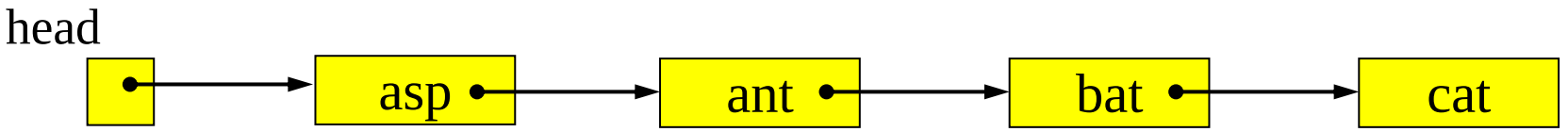
// 2) link the next field of newNode with the rest of the list

// 3) change head to point to the new node

```
Node newNode = new Node("asp", null);  // 1
newNode.next = head;                   // 2
head = newNode;                        // 3
```

Linked Lists vs. Arrays

- Size of linked list can be variable!
  - Arrays have fixed size.
- *Re-arrangement* of items in a linked list is (usually) *faster*.
- *Access* to elements is *slower* in a LL.



//********* demo 3 **Insert a Node** "rat" in **middle**

// 1) make nodeBefore point to the node before insertion point

// 2) create a new node with desired data

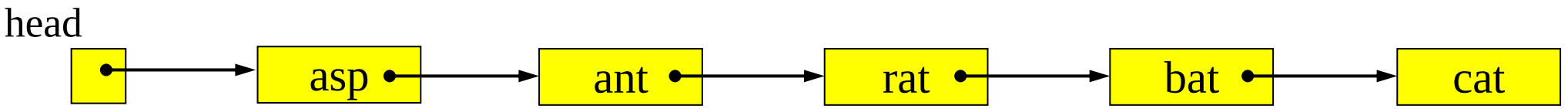
// 3) set newNode.next to refer to the items after nodeBefore

// 4) set nodeBefore.next to refer to newNode

```
Node nodeBefore = head.next;          //  1
newNode = new Node("rat",null);       //  2
newNode.next = nodeBefore.next;       //  3
nodeBefore.next = newNode;            //  4
```

Pitfalls with Pointers

- You should be aware that programming with references is very powerful, but can be tricky.
- **Aliasing**: `If two variables contain references to the same object, the state of the object can be modified using one variable's reference to the object, and then the altered state can be observed through the reference in the other variable.' (Gosling, Joy, Steele, *The Java Language Specification*).

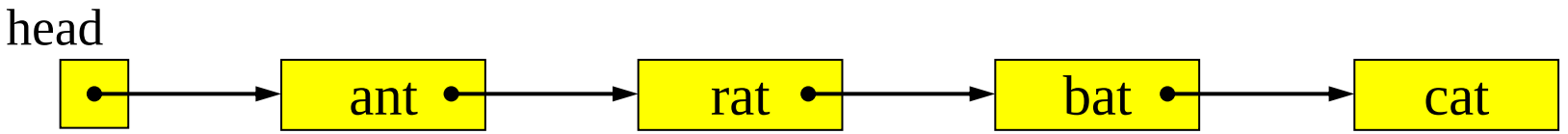


//********* demo 4 **Remove** the Node at the **front**

// 1) advance head to the next node in line

// 2) unreferenced node will be garbage collected

head = head.next; // 1



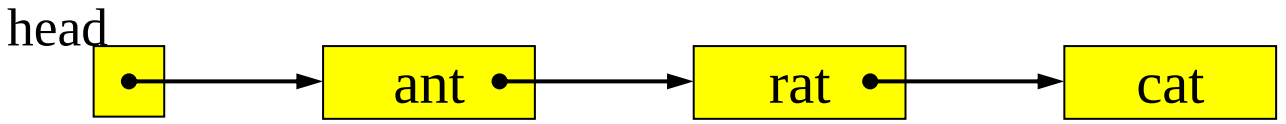
//********* demo 5 **Remove** the Node "bat" from the **middle**

// 1) make nodeBefore point to the node before the deletion point

// 2) set nodeBefore.next to the list following the deletion node

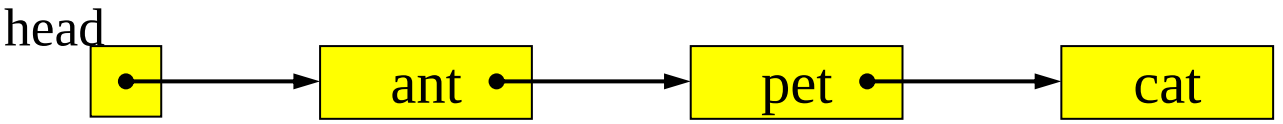
// 3) the unreferenced node will be garbage collected

```
nodeBefore = head.next;                          // 1
nodeBefore.next = nodeBefore.next.next; // 2
```



//********* demo 6 **Change** a Node's **data**, "rat" to "pet”

head.next.data = "pet";




```
//*********  demo 7 add and remove a Node after ant
      //  first, add a new node after ant     ant->trap->pet->cat->null
      //    follow steps in demo3

      System.out.println("7) ");
                                                  ← code here

      printList(head);
      // then remove the node after ant     ant->pet->cat->null
      //    follow steps in demo4
                                                  ← code here

      printList(head);
```

Null

- The last node of a linked list is a reference, but it is the **null reference** that refers to nothing!
- If some operation tries to use the object that the null ref. points to then an exception is raised (in Java NullPointerException).
- Not always easy to ensure all of these are caught.
- `I call it my billion-dollar mistake. It was the invention of the null reference in 1965. … This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.' (Prof. Sir C.A.R. Hoare)

Java LinkedList

- List Interface:
  - http://download.oracle.com/javase/1.4.2/docs/api/java/util/List.html
- LinkedList class
  - http://download.oracle.com/javase/1.4.2/docs/api/java/util/LinkedList.html