I/O

- Saving/loading information: I/O streams
- I/O streams are sequences of characters associated with files.
- Stream operations:
  - `read`, `readLine` (`BufferedReader`)
  - `print`, `println` (`PrintWriter`)
- Protocol: open (associate stream with file), read/write, close.
- File format
- Saving/loading objects
- Encapsulate the save and load operation in an object
- Separation of concerns:
  - Saving/loading an individual object
  - Saving/loading an array of objects

I/O

```
import java.io.*;
class AgendaEntry {
  private String name;
  private long telephone;
  public AgendaEntry(String n, int t)
  {
    name = n;
    telephone = t;
  }
  public void save(PrintWriter file)
  {
    file.println(name+“   ”+telephone);
  }
  // Continues below
```

I/O

```
  public void load(BufferedReader file)
  throws EOFException, IOException
  {
    StringTokenizer tokenizer;
    String line = file.readLine();
    if (line == null)
      throw new EOFException();
    tokenizer = new StringTokenizer(line);
    name = tokenizer.nextToken();
    String tel = tokenizer.nextToken();
    try {
      telephone = Long.parseLong(tel);
    }
    catch (NumberFormatException e) {
      telephone = 18005550000;
    }
  }
} // End of AgendaEntry
```

I/O

```
class Agenda {
  pivate AgendaEntry[] list;
  //...
  public void add_entry(AgendaEntry e) { ... }
  // ...
  public void save(String file_name)
  {
    FileWriter fw = new FileWriter(file_name);
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter file = new PrintWriter(bw);
    for (int i = 0; i < list.length; i++) {
      if (list[i] != null)
        list[i].save(file);
    }
    file.close();
  }
  // Continues below
```

I/O

```
public void load(String file_name)
{
  int line_num = 0;
  try {
    FileReader fr = new FileReader(file_name);
    BufferedReader file = new BufferedReader(fr);
    try {
      while (true) {
        list[line_num] = new AgendaEntry();
        list[line_num].load(file);
        line++;
      }
    }
    catch (EOFException e) {}
    file.close();
  }
  catch (FileNotFoundException e) { ... }
  catch (IOException e) { ... }
}
```

Data structures

- Algorithms: procedures to solve problems involving information
- Data structures: organizing, handling and managing information
- A *data-structure* is an arrangement of data in a particular and well defined pattern of organization.
- Basic examples:
  - Variables (no structure)
  - Objects (structure given by aggregation)
  - Arrays (linear, list-like, structure)
- Collections: sets, lists, stacks, queues, trees, graphs, dictionaries, etc.
- A *set* is an unordered collection of elements, without repetitions. A *list* is an ordered collection of elements, possibly with repetitions.
- Homogeneous vs heterogeneous collections

Data structures

- Abstract Data Types (ADTs)
  - An ADT is a type representing a data-structure, with some operations on its elements.
  - Separating interface from implementation: A given ADT may be implemented using different underlying data-structures. For example, a *set* could be implemented as an array, a Vector, a *linked-list*, etc.

- A *dynamic data-structure* is a data-structure which can change.

- A *collection* is an ADT which supports operations for adding and removing elements (hence it is a dynamic data-structure.)

- A dynamic data-structure has a variable size, in contrast with an array or an object which have a fixed size.

- "Adding" to an array modifies the array, but it doesn't change its overall structure. "Growing" an array changes its overall structure.

The List ADT

- A list is an abstract data-type
- A list is a collection
- A list is a dynamic data-structure
- List operations:
  - Adding an element
  - Removing an element
  - Obtaining an element
  - Length
- Possible implementations
  - Arrays
  - Growing arrays
  - Vectors
  - Linked-lists

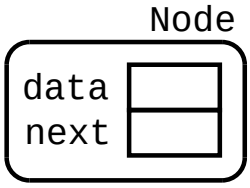
Linked Lists

- A *linked-list* is a dynamic data-structure consisting of a sequence of objects called *nodes*, where each node has a reference or link to the next node in the sequence.

- A linked-list is a collection.

- Nodes are a recursive data-structure

```
class Node {
  String data;
  Node next;
}
```



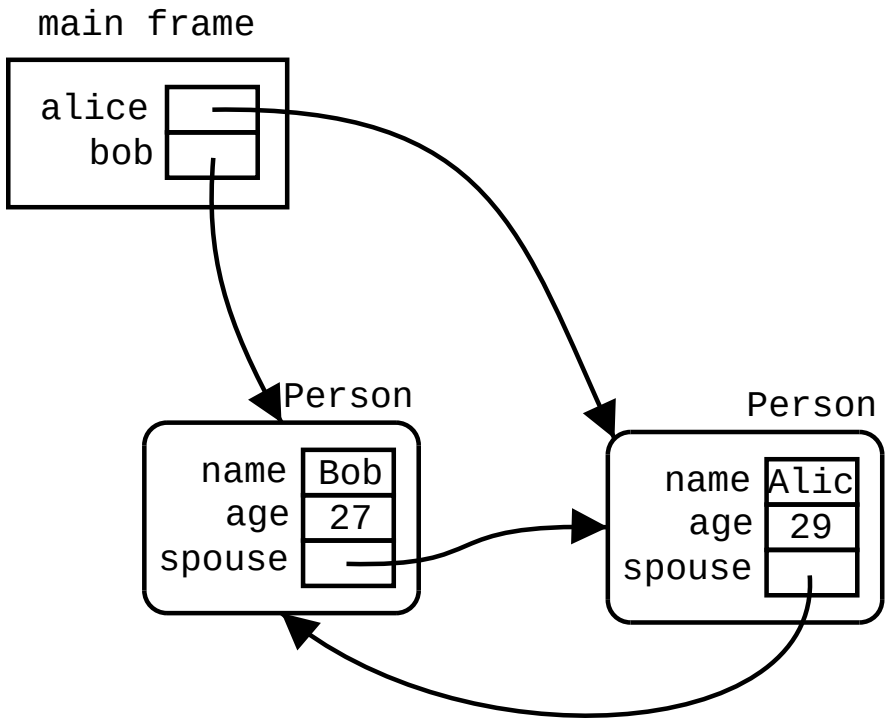
- A recursive data-structure has references to objects of its own type

Recursive data-structures

```
class Person {
  String name;
  int age;
  Person spouse;
  public Person(String n, int a)
  {
    name = n;
    age = a;
    spouse = null;
  }
  public void marry(Person p)
  {
    spouse = p;
    p.spouse = this;
  }
}
```

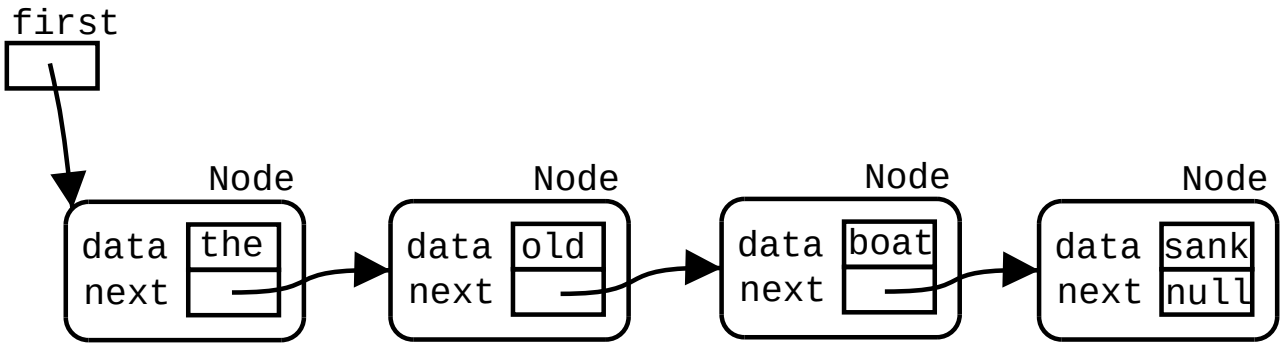
Recursive data-structures

```
public class Marriage {
  public static void main(String[] args)
  {
    Person alice = new Person("Alice", 29);
    Person bob = new Person("Bob", 27);
    alice.marry(bob);
  }
}
```



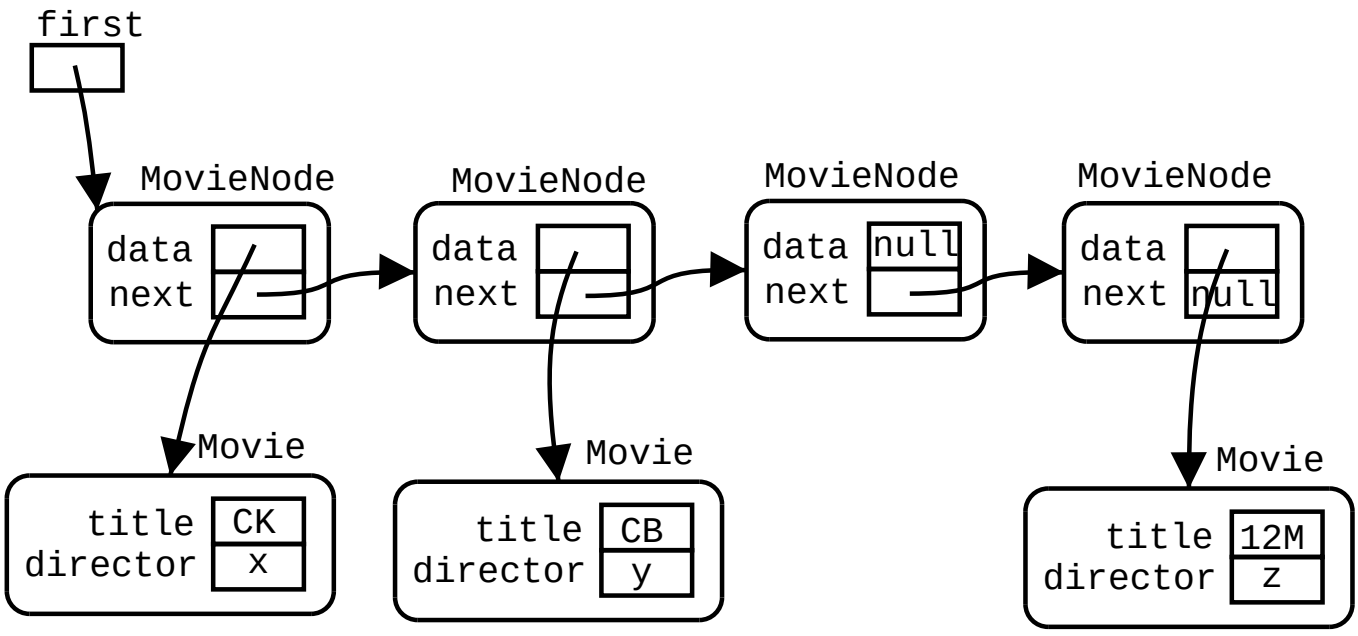
Linked Lists

```
class Node {
  String data;
  Node next;
  void set_data(String d) { data = d; }
  String get_data() { returns data; }
  void set_next(Node n) { next = n; }
  Node get_next() { return next; }
}
```



Linked Lists

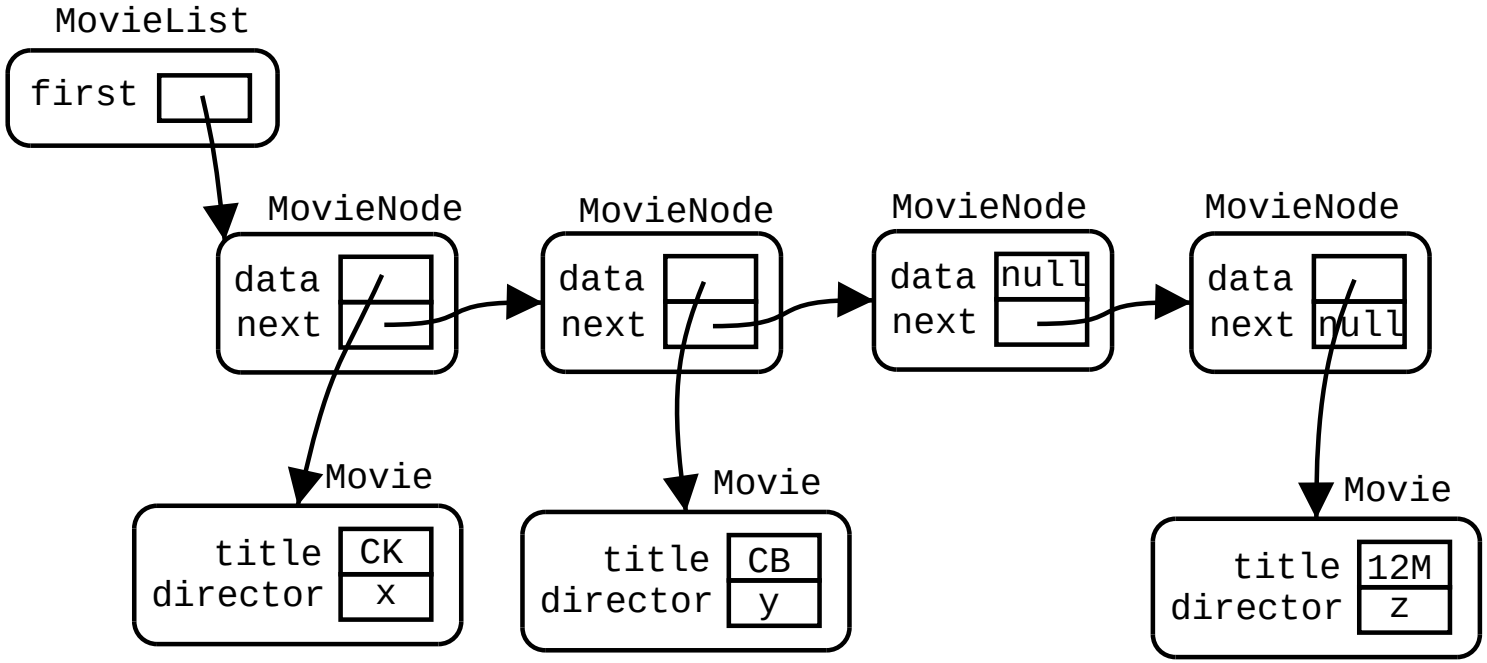
```
class Movie {
  String title, director;
  // ...
}

class MovieNode {
  Movie data;
  MovieNode next;
}
```



Linked Lists

```
class MovieNode {
  private Movie data;
  private MovieNode next;

  public MovieNode(Movie m, MovieNode n) {
    data = m;
    next = n;
  }
  public Movie get_movie() { return data; }
  public MovieNode get_next() { return next; }
  public void set_movie(Movie m)
  {
    data = m;
  }
  public void set_next(MovieNode n)
  {
    next = n;
  }
}
```

Linked Lists



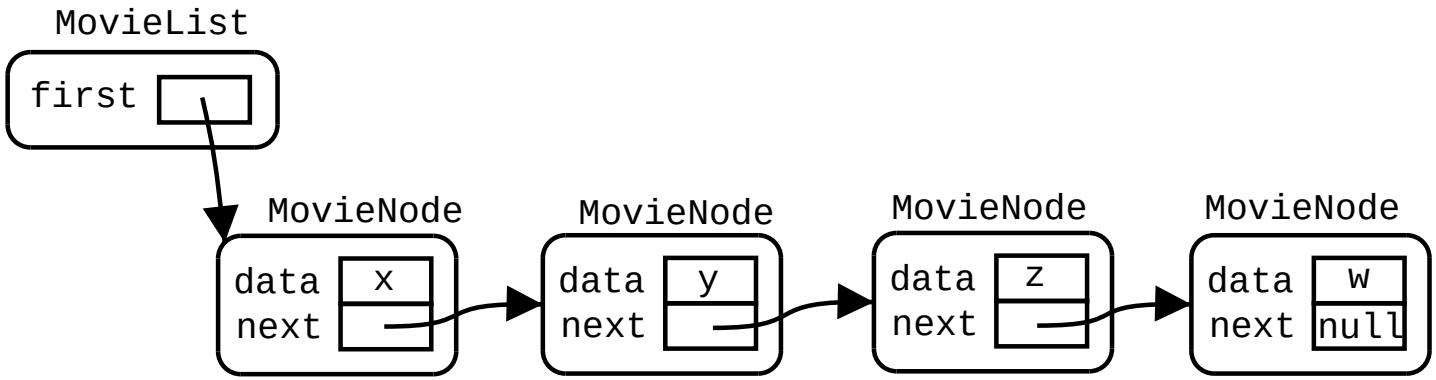
Linked Lists

```
class MovieList {
  private MovieNode first;

  public MovieList() { first = null; }

  public void add(Movie m)
  {
    MovieNode new_node = new MovieNode(m, first);
    first = new_node;
  }
}
```

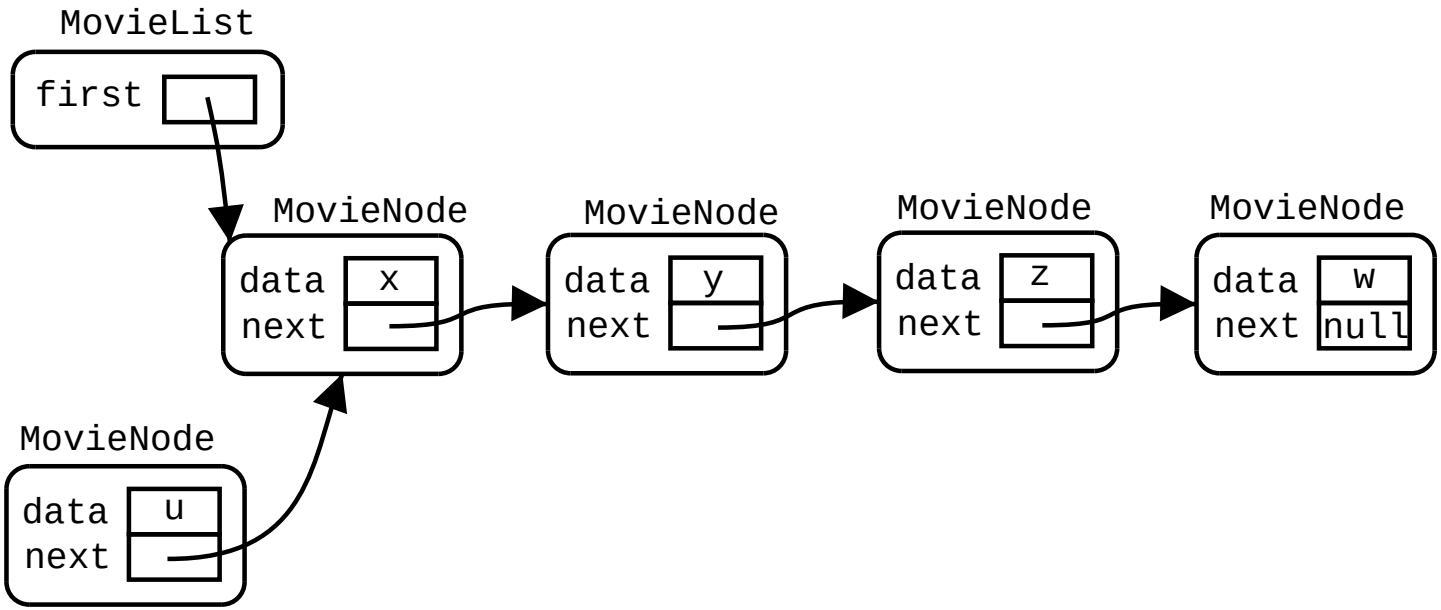
Linked Lists

```
class Test {
  public static void main(String[] args)
  {
    MovieList l = new MovieList();
    Movie w = new Movie(“abc”,“def”);
    Movie x = new Movie(“bca”,“efd”);
    Movie z = new Movie(“cba”,“fef”);
    Movie y = new Movie(“xxx”,“yyy”);
    l.add(w);
    l.add(z);
    l.add(y);
    l.add(x);
    Movie u = new Movie(“fed”,“bac”);
    l.add(u);
  }
}
```

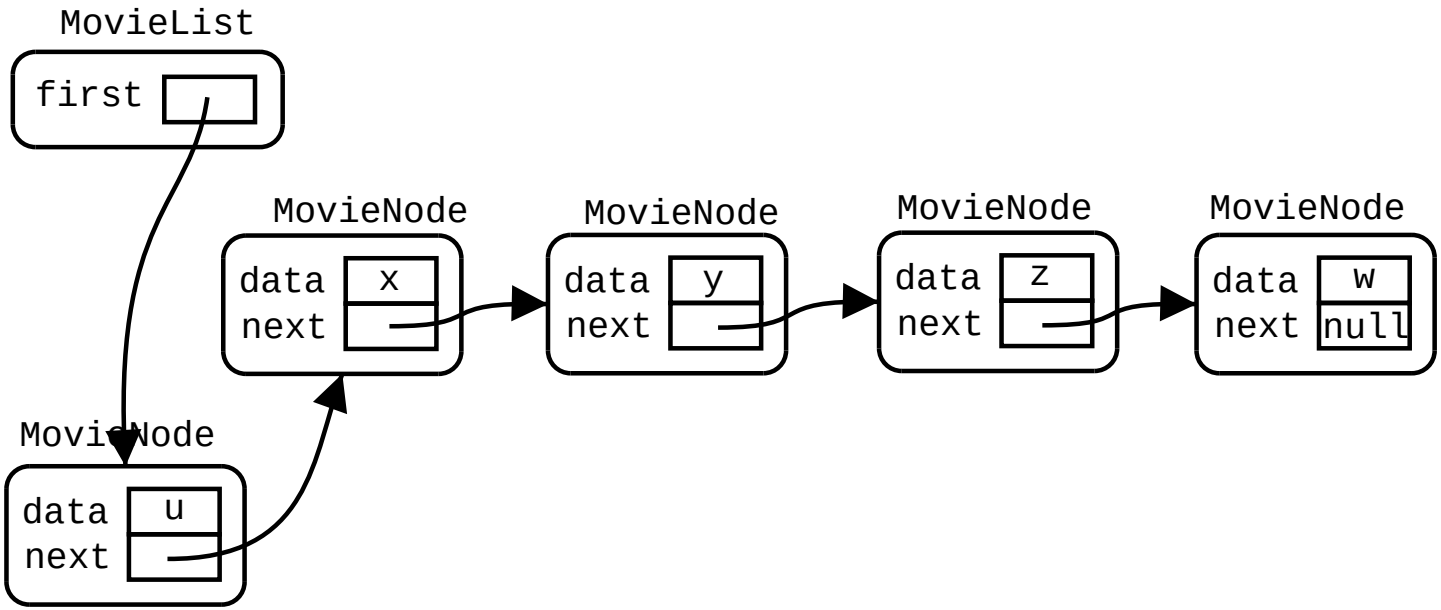
Linked Lists



Linked Lists



Linked Lists



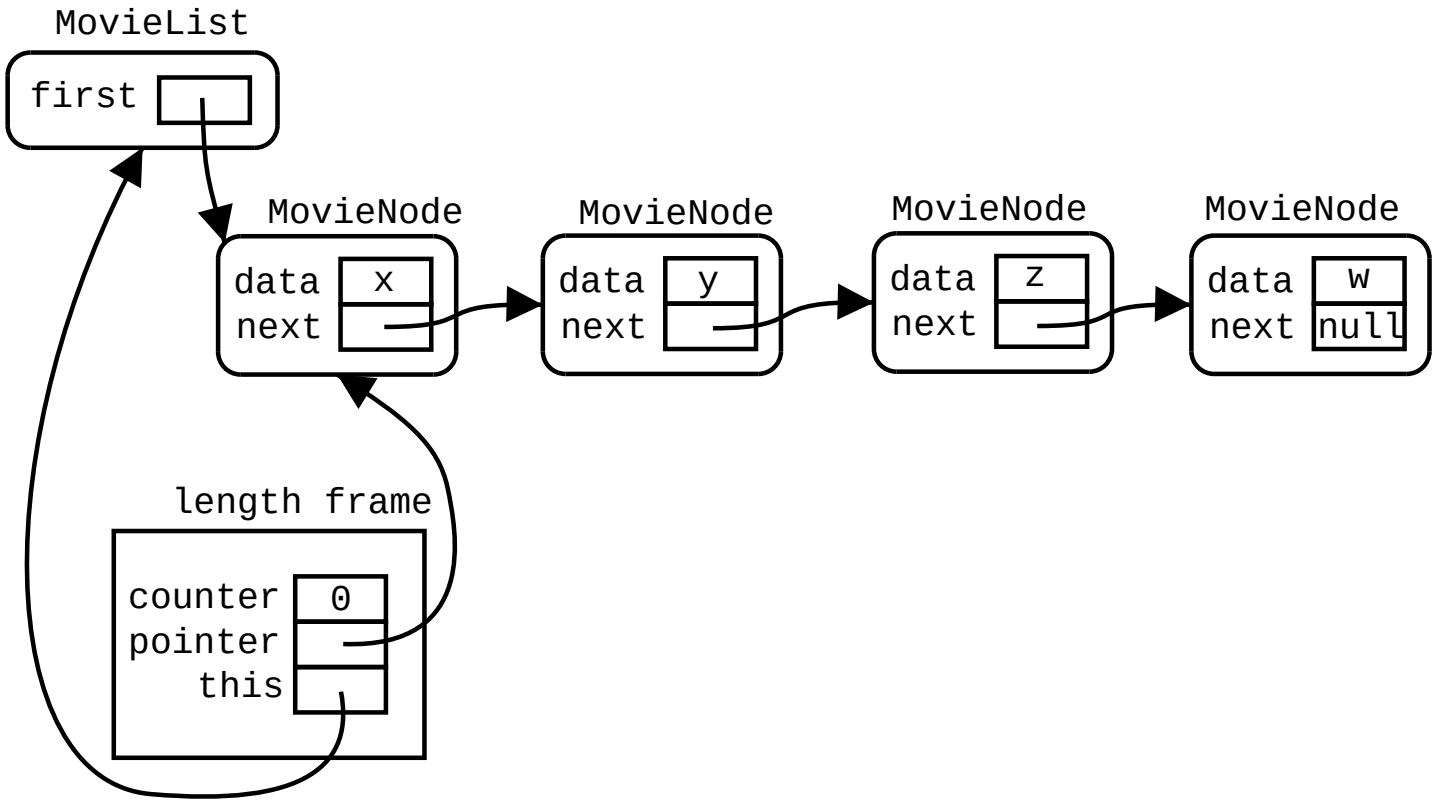
Linked Lists

```
class MovieList {
  private MovieNode first;
  //...
  public int length()
  {
    int counter = 0;
    MovieNode pointer = first;
    while (pointer != null) {
      pointer = pointer.get_next();
      counter++;
    }
    return counter;
  }
}
```

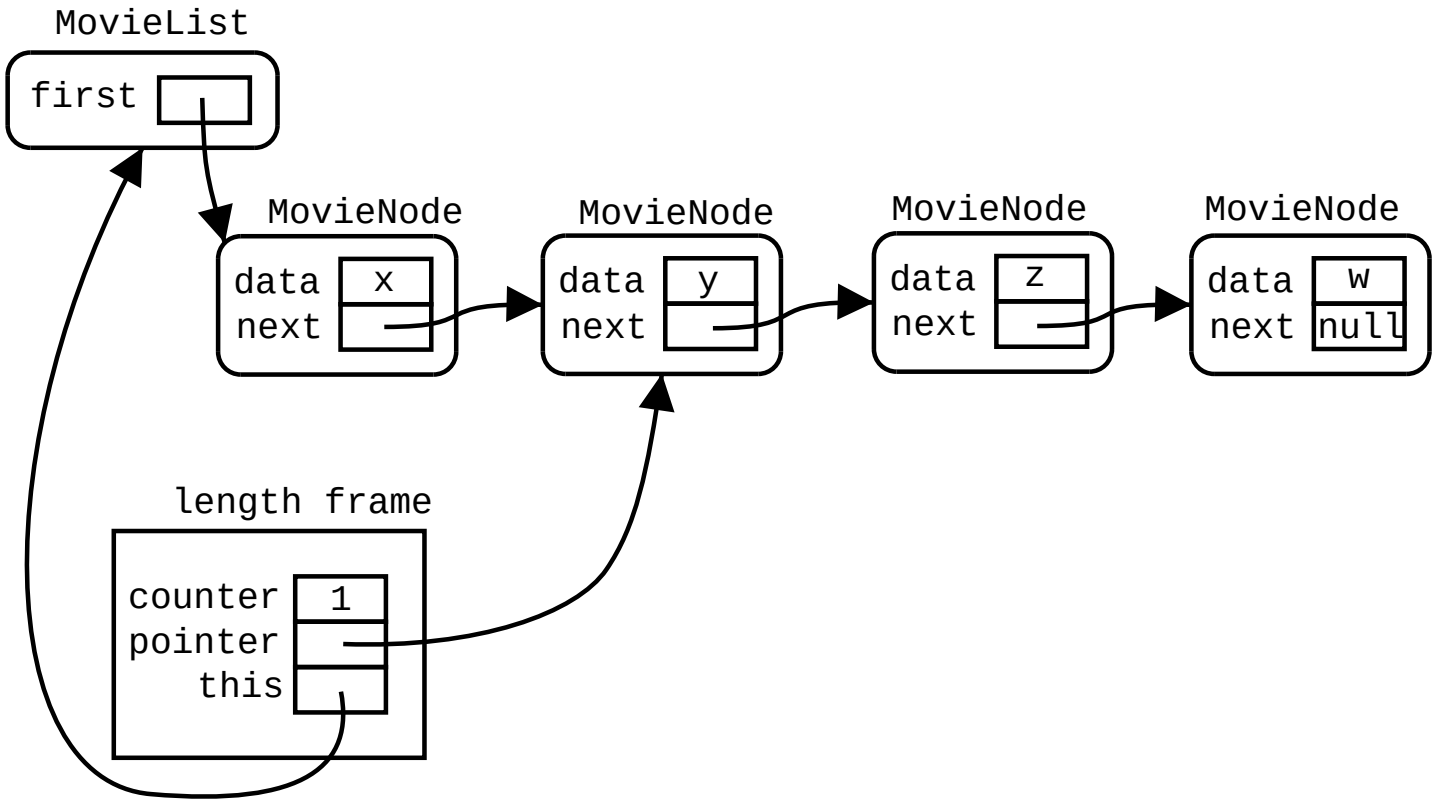
Linked Lists

```
class Test {
  public static void main(String[] args)
  {
    MovieList l = new MovieList();
    Movie w = new Movie(“abc”,“def”);
    Movie x = new Movie(“bca”,“efd”);
    Movie z = new Movie(“cba”,“fef”);
    Movie y = new Movie(“xxx”,“yyy”);
    l.add(w);
    l.add(z);
    l.add(y);
    l.add(x);
    int s = l.length();
  }
}
```

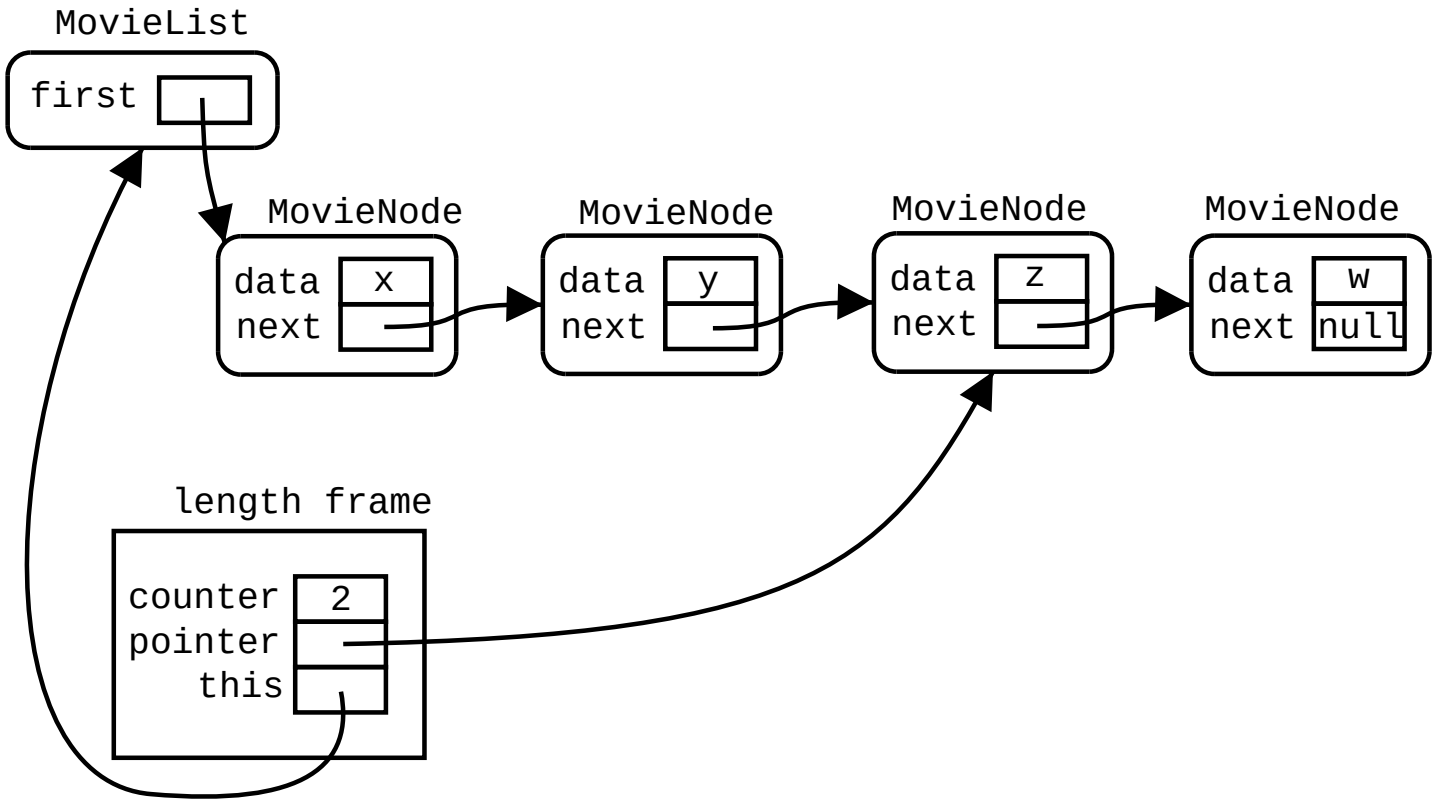
Linked Lists



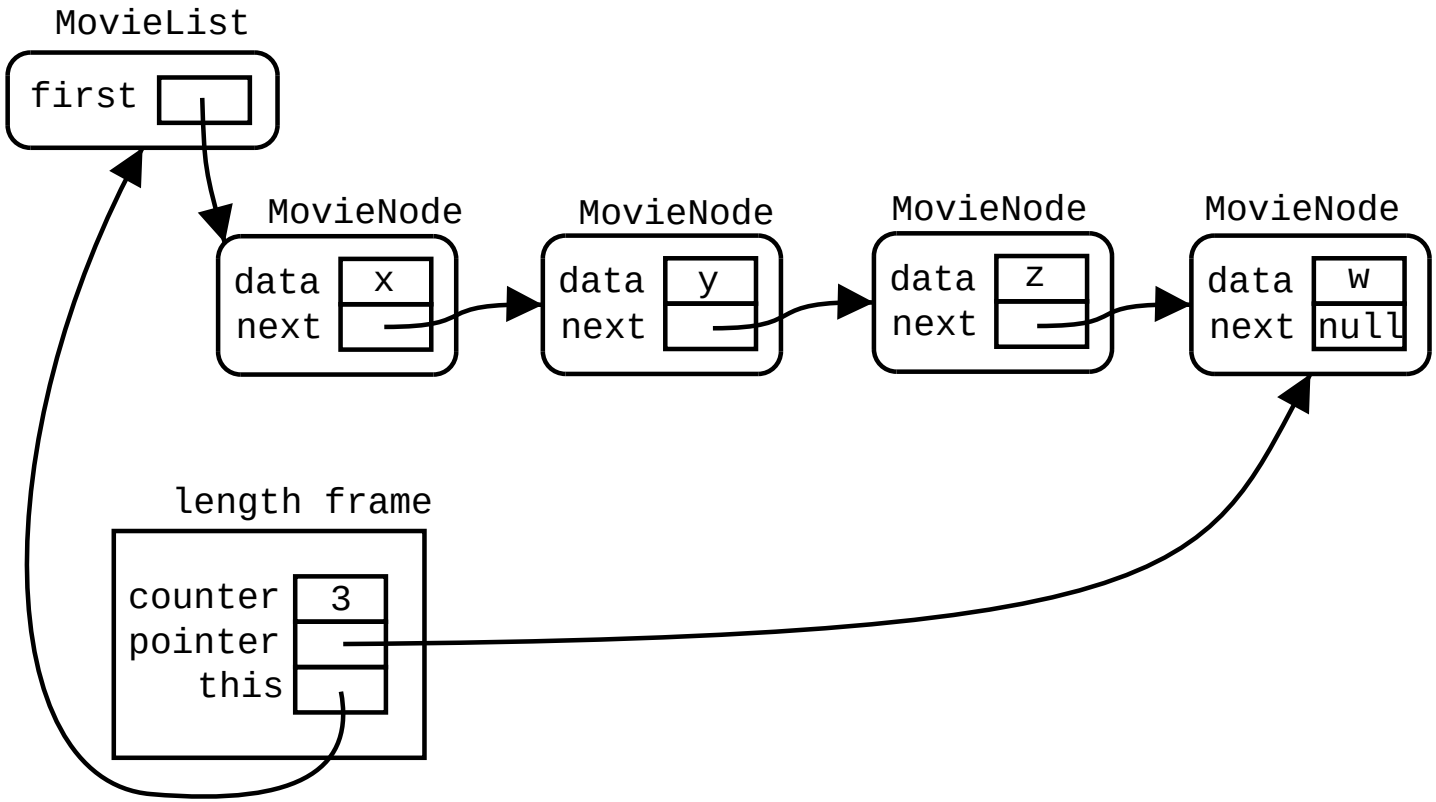
Linked Lists



Linked Lists



Linked Lists



Linked Lists