Collections and Data-Structures

- Programs manipulate information
- Information can be complex
- Information needs to be stored and organized somehow
- A *collection* is an object that stores other objects
- Operations on collections
  - Adding elements
  - Removing elements
  - Finding/Retreiving elements
  - etc...

Collections and Data-Structures

```
public class Set
{
  //...
  public Set() { ... }
  public void add(Object o) { ... }
  public void remove(Object o) { ... }
  public boolean isMember(Object o) { ... }
  public int size() { ... }
}
```

Collections and Data-Structures

- A collection can be seen as an *Abstract Data Type* (ADT), this is:
  - It allows certain operations,
  - but the implementation of these operations is hidden from the client,
  - so, several possible *underlying implementations* are possible,

Collections and Data-Structures

```
public interface Set
{
  public void add(Object o);
  public void remove(Object o);
  public boolean isMember(Object o);
  public int size();
}
```

Collections and Data-Structures

```
public class ArraySet implements Set
{
  private Object[] array;
  private int count;

  public Set()
  {
    array = new Object[1000];
    count = 0;
  }
  public void add(Object o) { ... }
  public void remove(Object o) { ... }
  public boolean isMember(Object o) { ... }
  public int size() { ... }
}
```

Collections and Data-Structures

```
class Brick { ... }
class Wall
{
  Set s;
  void build()
  {
    s = new ArraySet();
    s.add(new Brick());
    s.add(new Brick());
    //...
  }
}
```

Collections and Data-Structures

- The implementation of a collection relies on a particular *data-structure*.
- A *data-structure* is an arrangement of information in a particular pattern
- Kinds of data-structures
  - Linear: arrays, linked-lists, ...
  - Non-linear: trees, graphs, hash-tables...
- Data-structures support particular operations

Collections and Data-Structures

- Some important linear ADTs
  - Lists
  - Stacks
  - Queues

- Some important lon-linear ADTs
  - Sets
  - Bags
  - Trees
  - Graphs
  - Dictionaries (maps)
  - ...

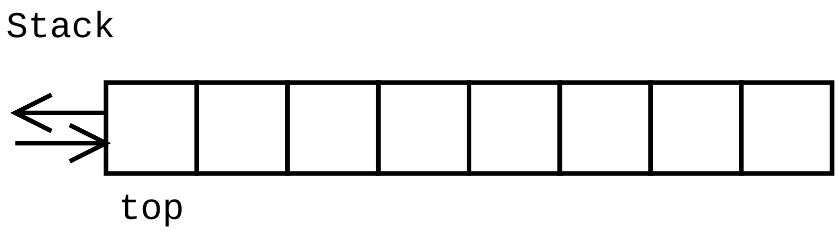
The List ADT

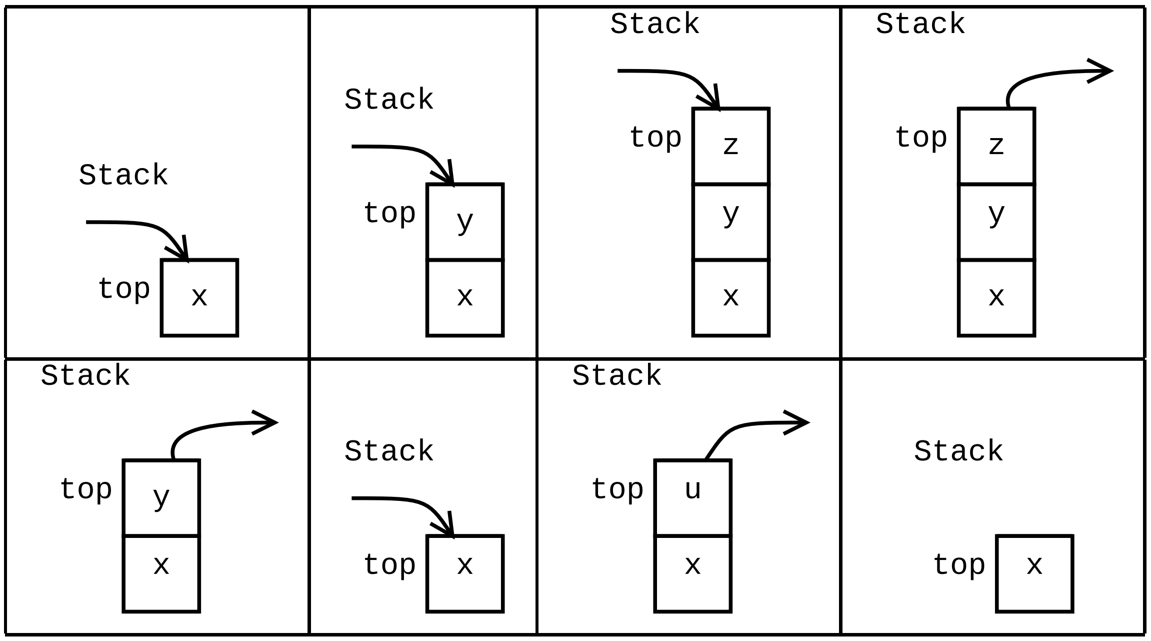
- List operations:
  - Adding an element
  - Removing an element
  - Obtaining an element
  - Length
  - Finding an element
  - etc.
- Possible implementations
  - Arrays
  - Growing arrays
  - Vectors
  - Linked-lists

The Stack ADT

- A *stack* (LIFO, or FILO) is a linear collection with (at least) the following operations:
  - *push*: adds an item at the "top" of the sequence
  - *pop*: removes the "top" item of the sequence
  - *top*: returns the top item without removing it
  - *isempty*: returns true if the sequence has no items

Stacks

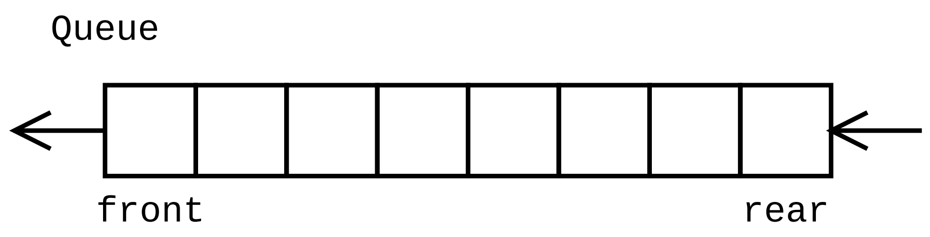

Stack
top

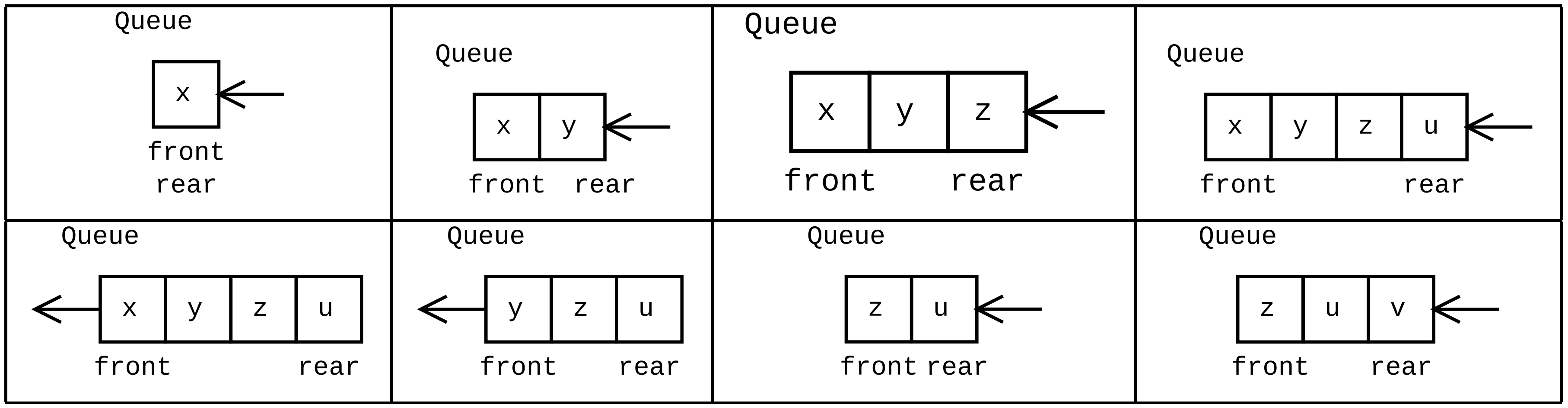

Stack
top
x
Stack
top
y
x
Stack
top
z
y
x
Stack
top
z
y
x
Stack
top
y
x
Stack
top
x
Stack
top
u
x
Stack
top
x

The Queue ADT

- A *queue* (FIFO) is a linear collection with (at least) the following operations:
  - *enqueue*: adds an item at the end of the sequence
  - *dequeue*: removes the first item of the sequence
  - *peek*: gets the first item of the sequence without removing it
  - *isempty*: returns true if the sequence has no items

Queues


Queue
front
rear


Queue
x
front
rear
Queue
x
y
front
rear
Queue
x
y
z
front
rear
Queue
x
y
z
u
front
rear
Queue
x
y
z
u
front
rear
Queue
y
z
u
front
rear
Queue
z
u
front
rear
Queue
z
u
v
front
rear

Implementing Stacks

```
public interface Stack
{
  public void push(Object obj);
  public void pop();
  public Object top();
  public boolean isEmpty();
}
```

Implementing Stacks

```
class ArrayStack implements Stack {
  private Object[] list;
  private int top;

  public Stack()
  {
    list = new Object[1000];
    top = 0;
  }
  public void push(Object obj)
  {
    if (top >= list.length)
      grow_array(100);
    list[top] = obj;
    top++;
  }
```

McGill

Implementing Stacks

```
  public void pop()
  {
    top--;
  }
  public Object top()
  {
    return list[top];
  }
  public boolean isEmpty()
  {
    return top == 0;
  }
  private void grow_array(int n)
  {
    ...
  }
}  // End of Stack
```

Implementing Queues

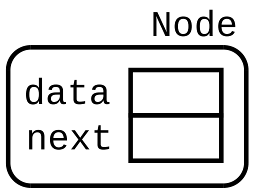
```
public interface Queue
{
  public void enqueue(Object obj);
  public void dequeue();
  public Object peek();
  public boolean isEmpty();
}
```

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.

- Nodes are a recursive data-structure

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



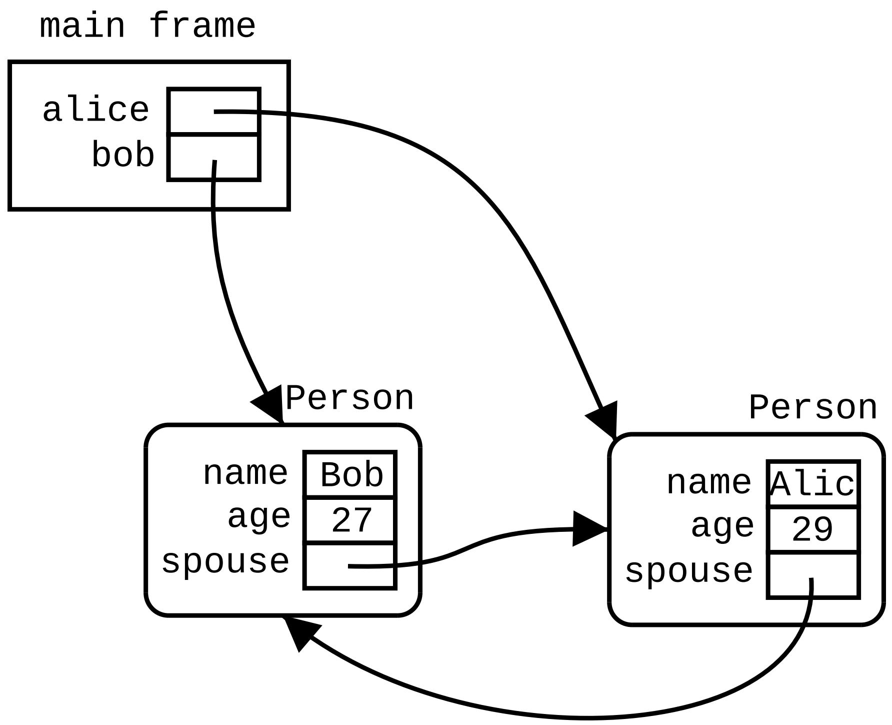
- A recursive data-structure has references to objects of its own type

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;
  }
}
```

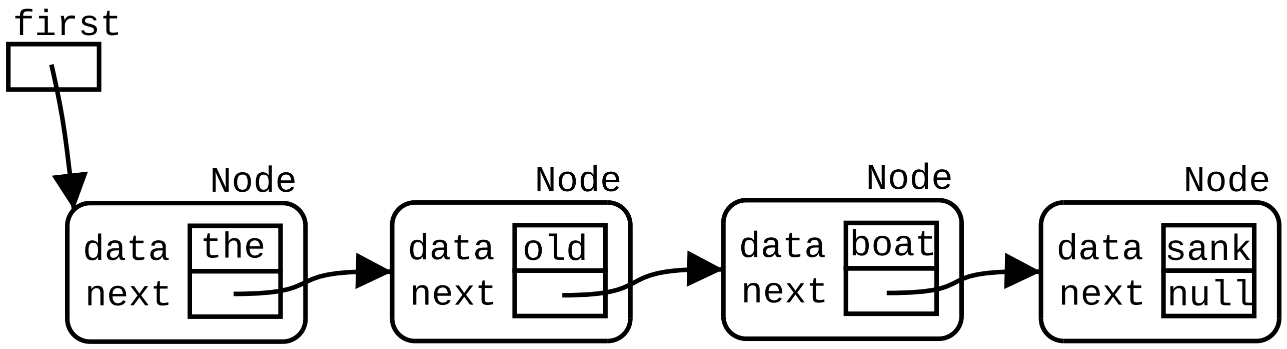
Recursive data-structures

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



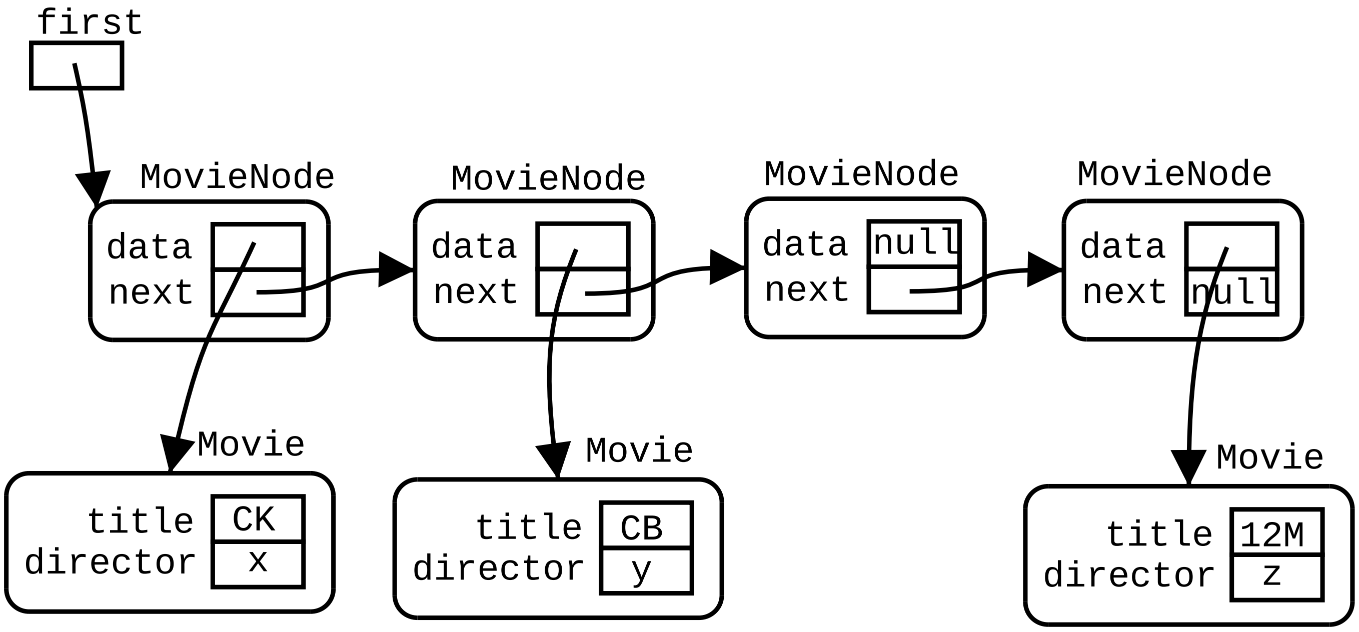
Linked Lists

```
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

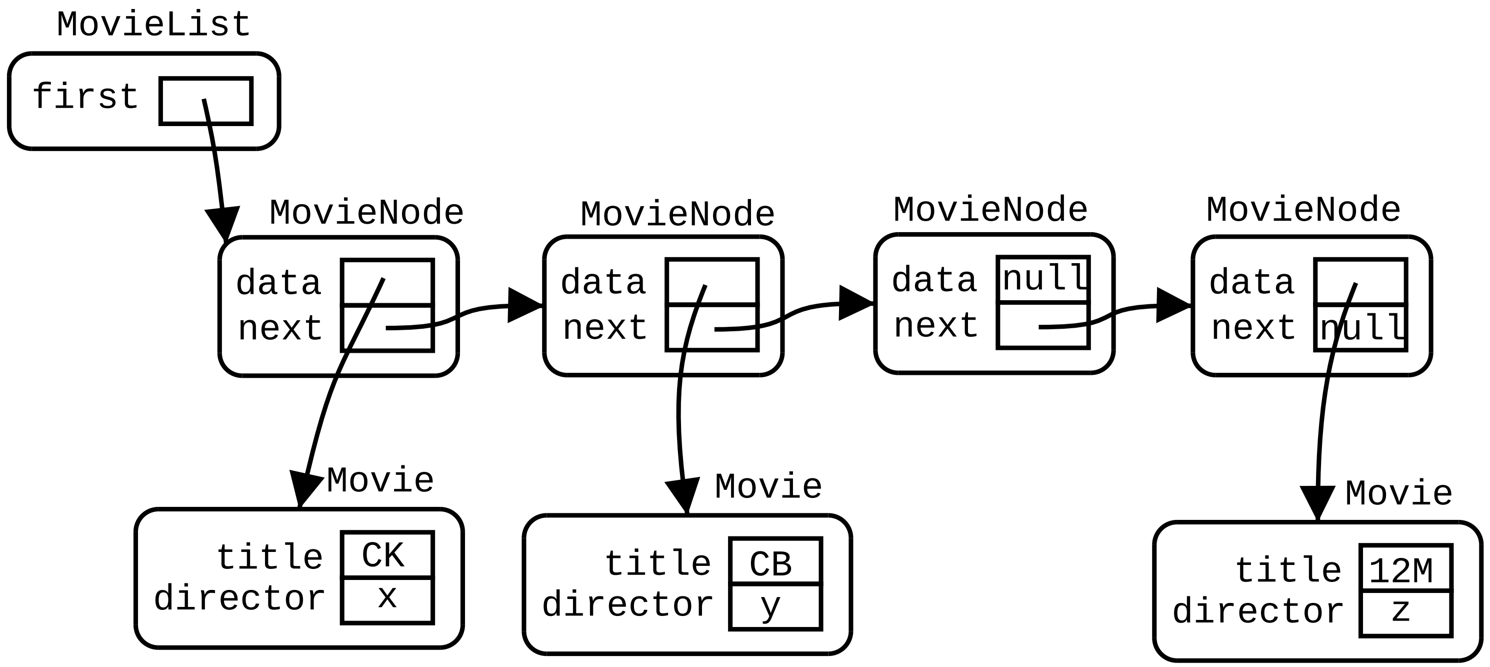
```
class Movie {
  String title, director;
  // ...
}

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



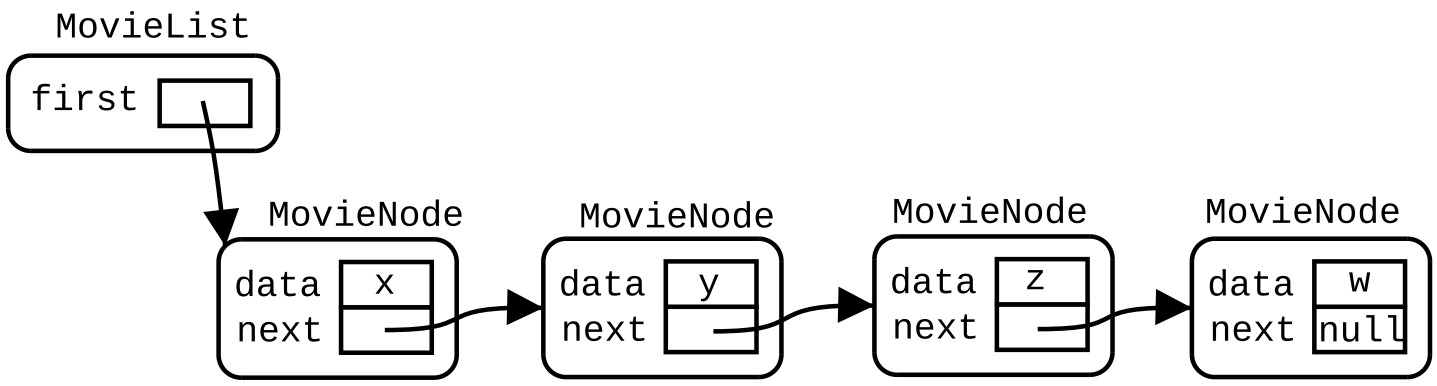
Linked Lists

```
class MovieList {
  private MovieNode first;

  public MovieList() { first = null; }

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

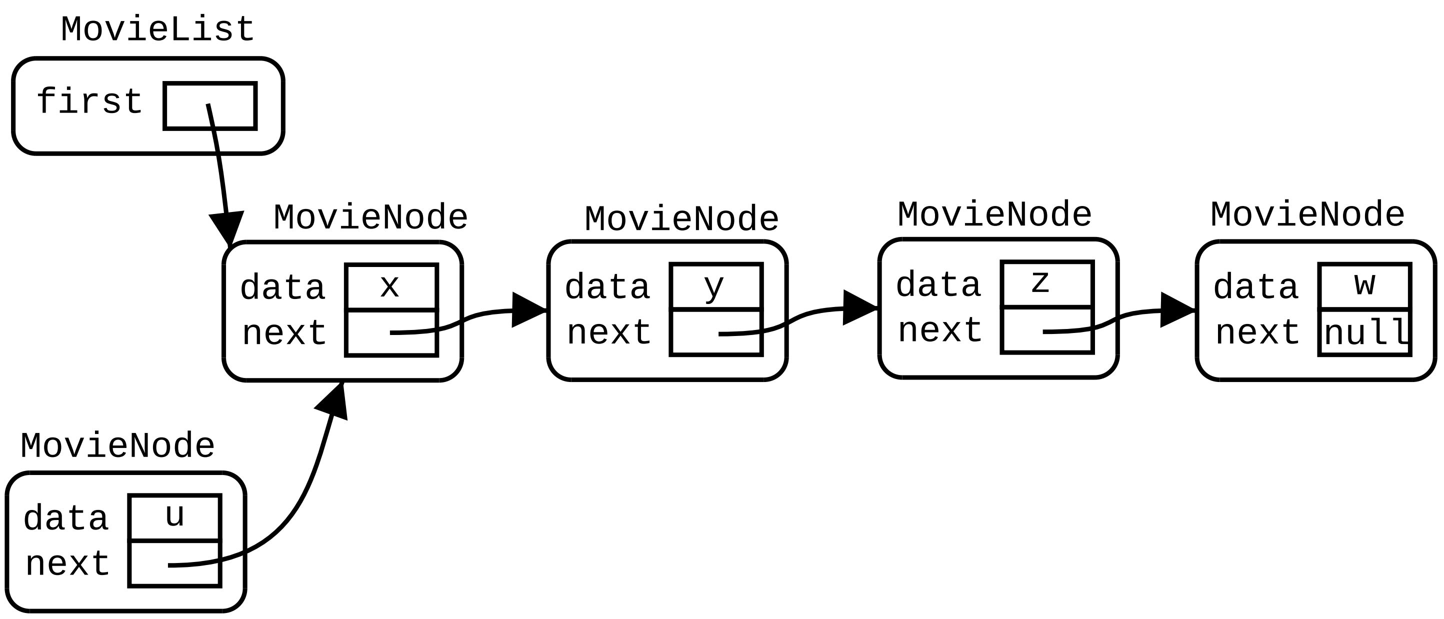
Linked Lists

```
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);
  }
}
```

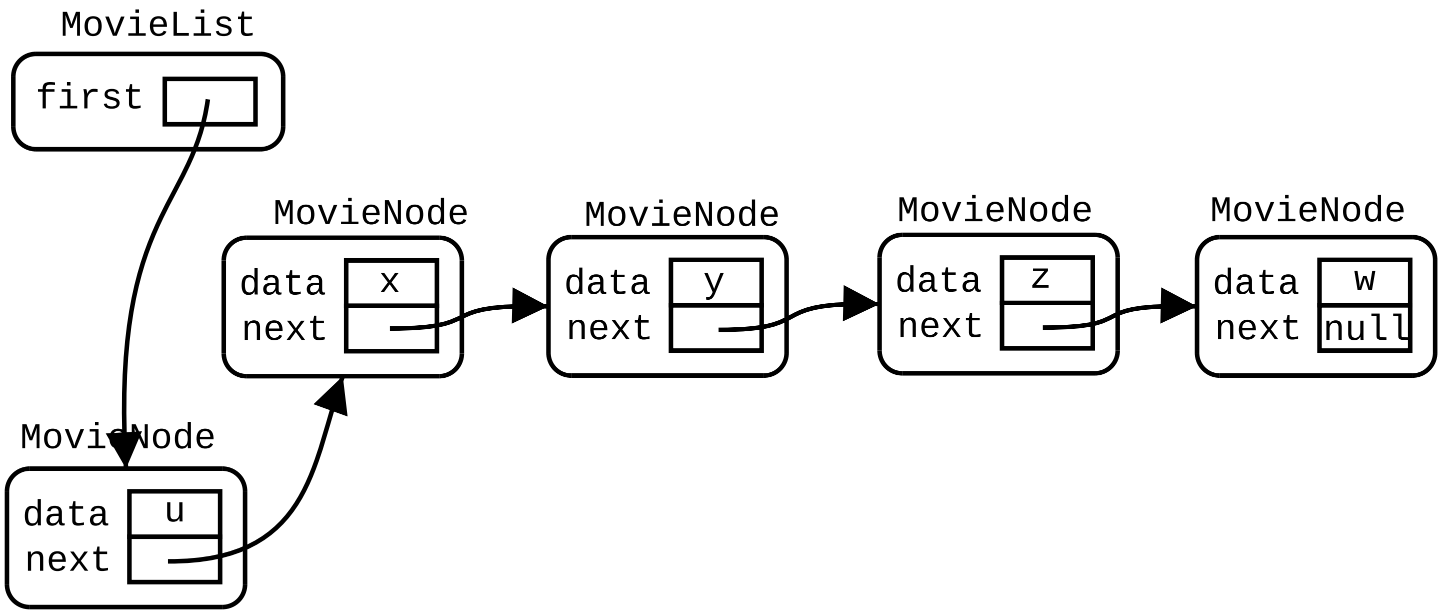
Linked Lists



Linked Lists



Linked Lists



The end