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

- 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

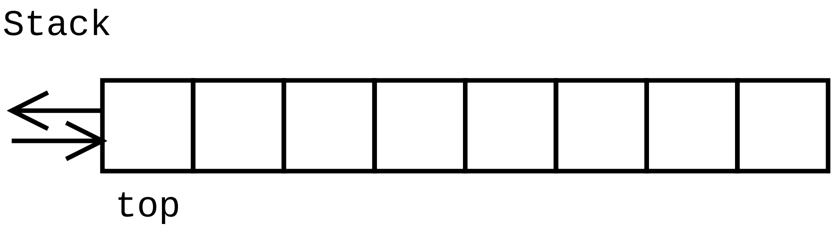
Collections and Data-Structures

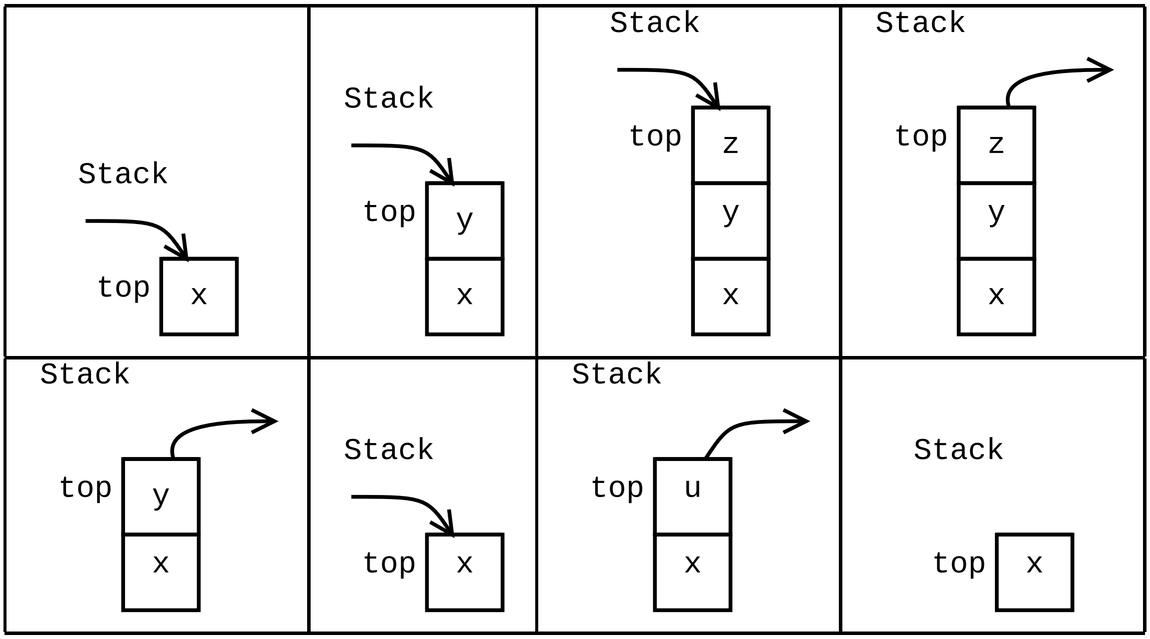
- Some important linear ADTs
  - Lists
  - Stacks
  - Queues
- Some important non-linear ADTs
  - Sets
  - Bags
  - Trees
  - Graphs
  - Dictionaries (maps)
  - ...

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

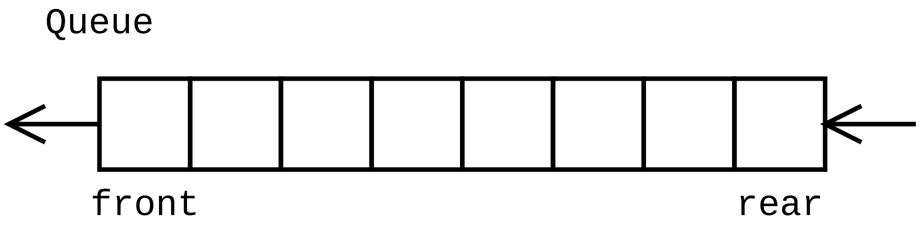

Stack
top

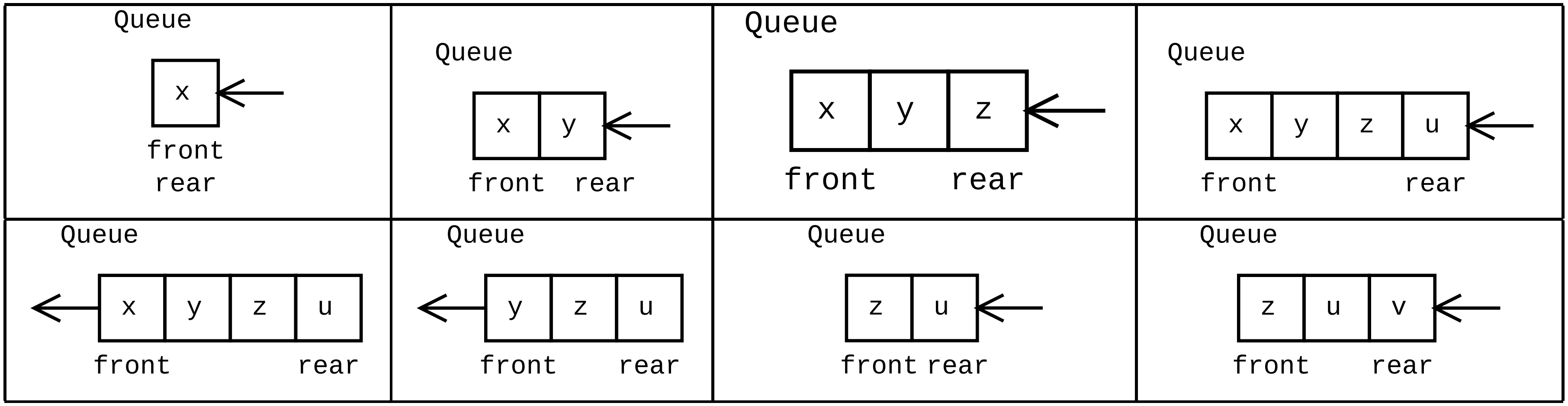

Stack
top
x
Stack
top
y
x
Stack
top
z
y
x
Stack
top
z
y
x
Stack
top
y
x
Stack
top
x
Stack
top
u
x
Stack
top
x

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

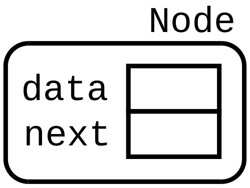

Queue
x
front
rear
Queue
x
y
front
rear
Queue
x
y
z
front
rear
Queue
x
y
z
u
front
rear
Queue
x
y
z
u
front
rear
Queue
y
z
u
front
rear
Queue
z
u
front
rear
Queue
z
u
v
front
rear

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

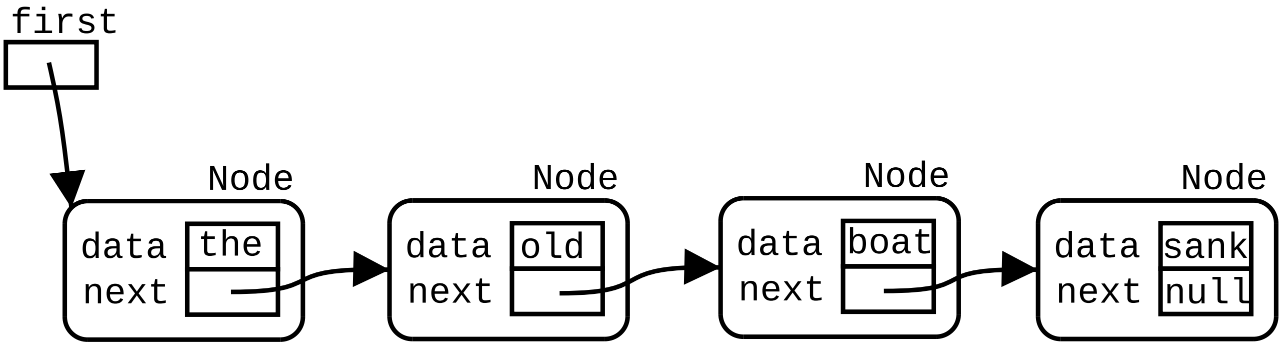
```
class Node {
  String data;
  Node next;
}
```



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

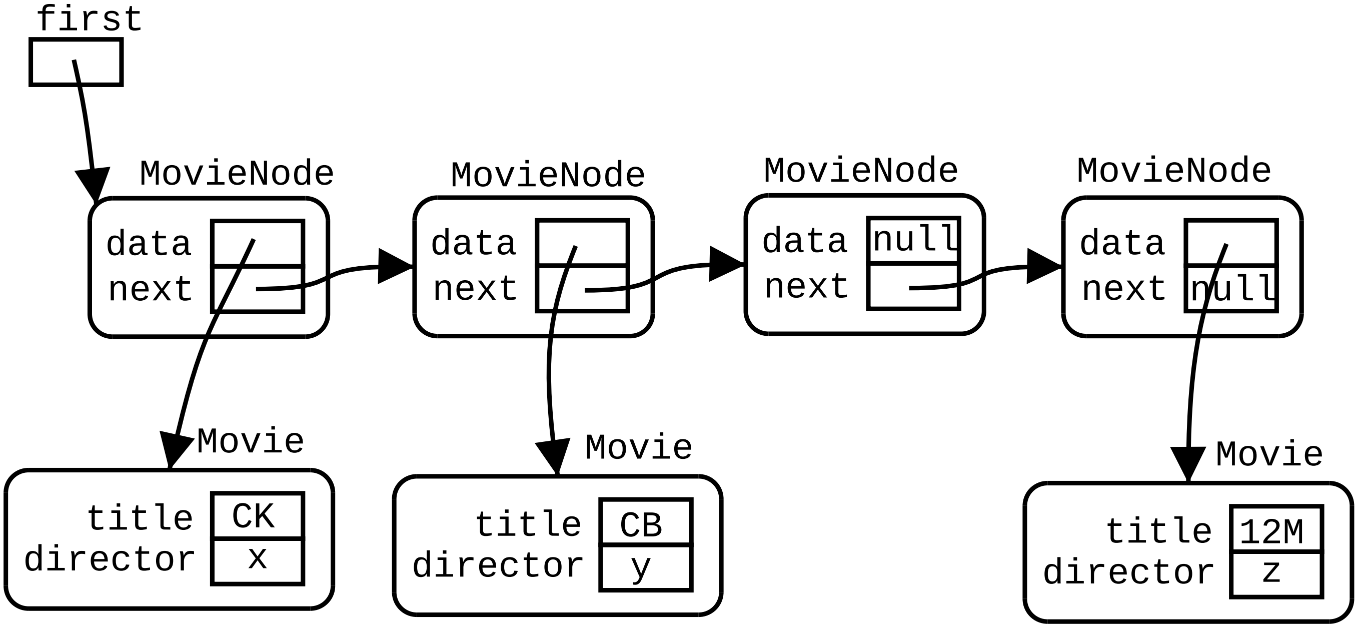
Linked Lists

```
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;
  }
}
```

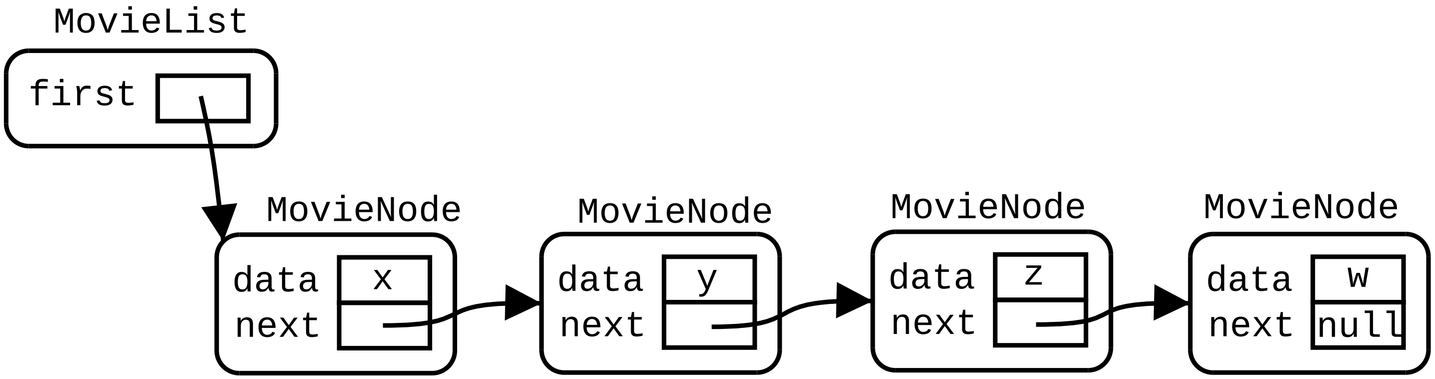
Linked Lists

```
class MovieList {
  private MovieNode first;

  public MovieList() { first = null; }

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

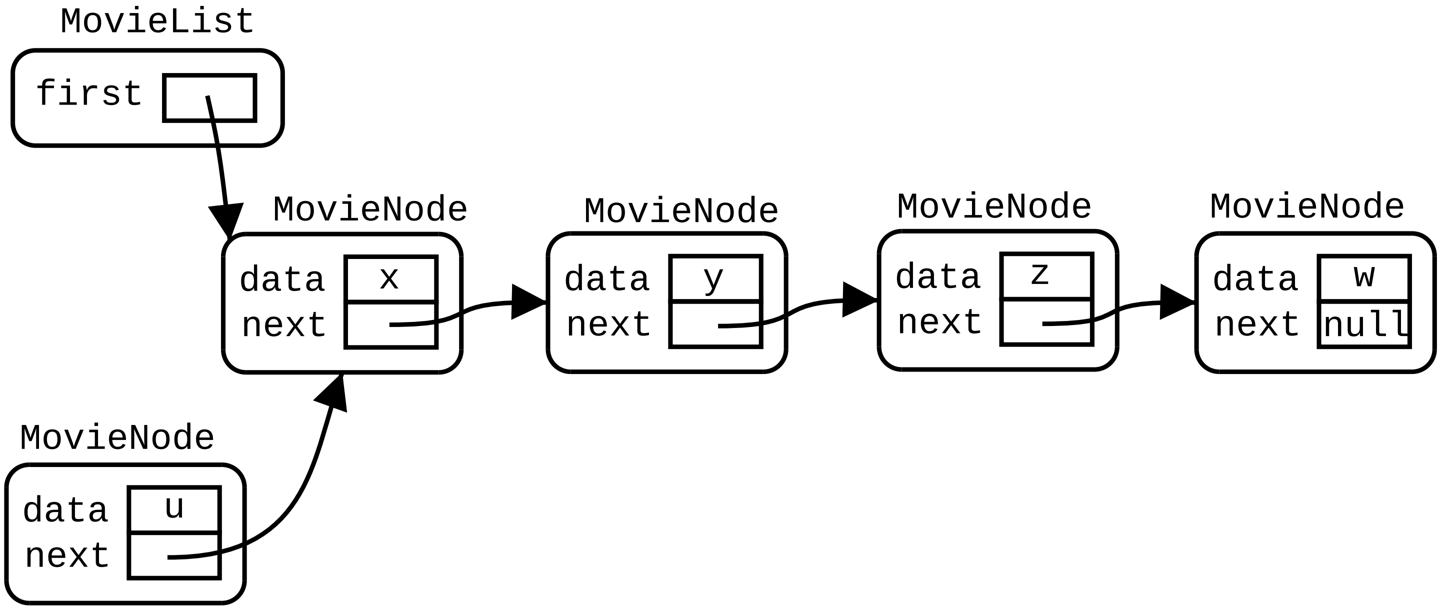
Linked Lists

```
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);
  }
}
```

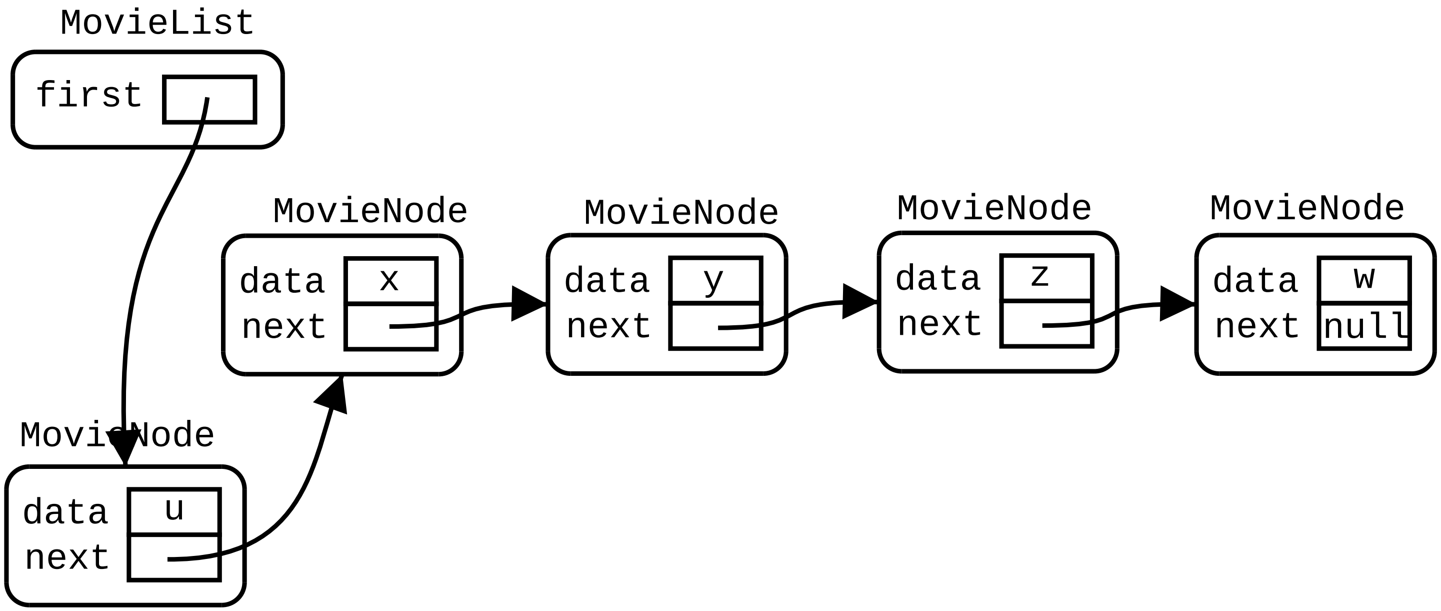
Linked Lists



Linked Lists



Linked Lists



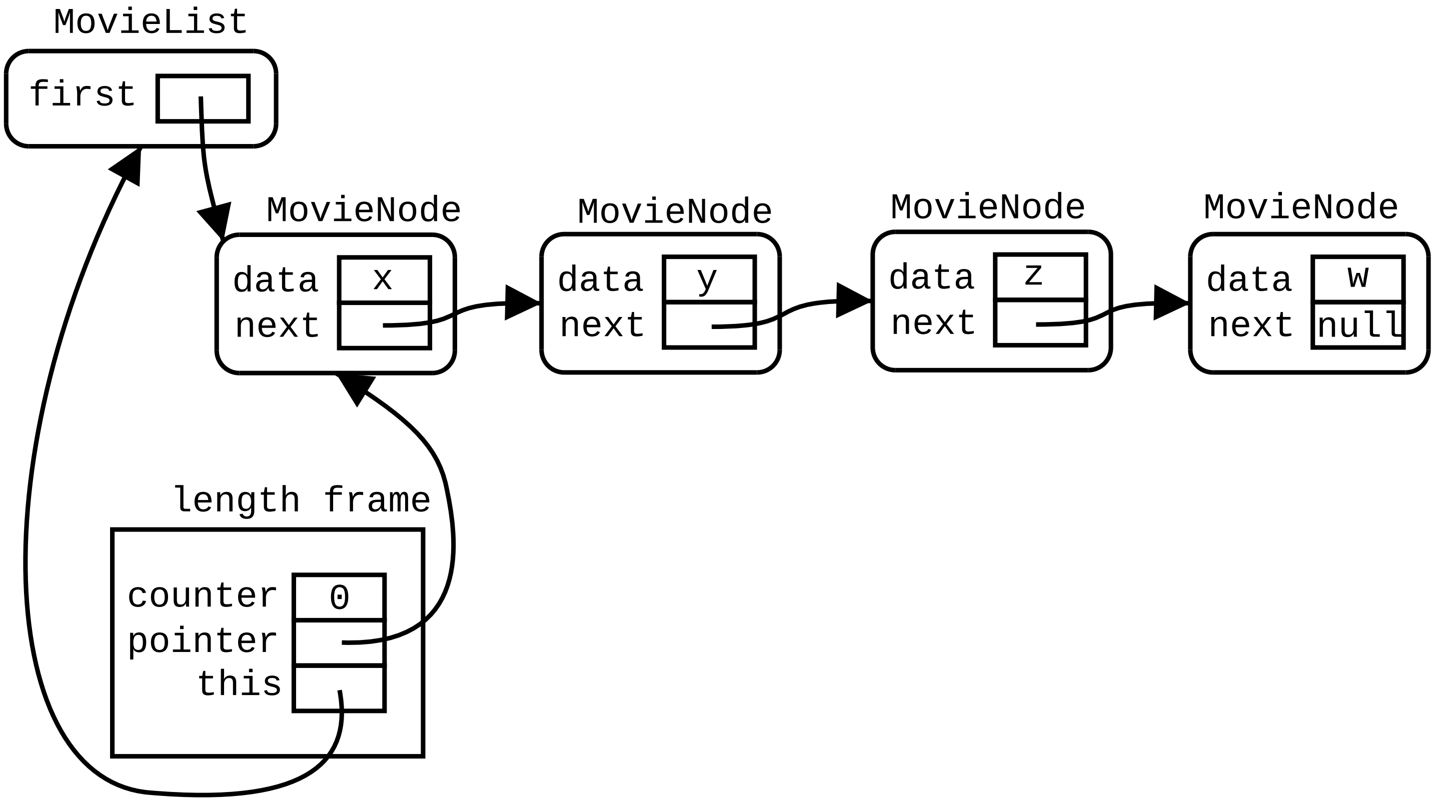
Linked Lists

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

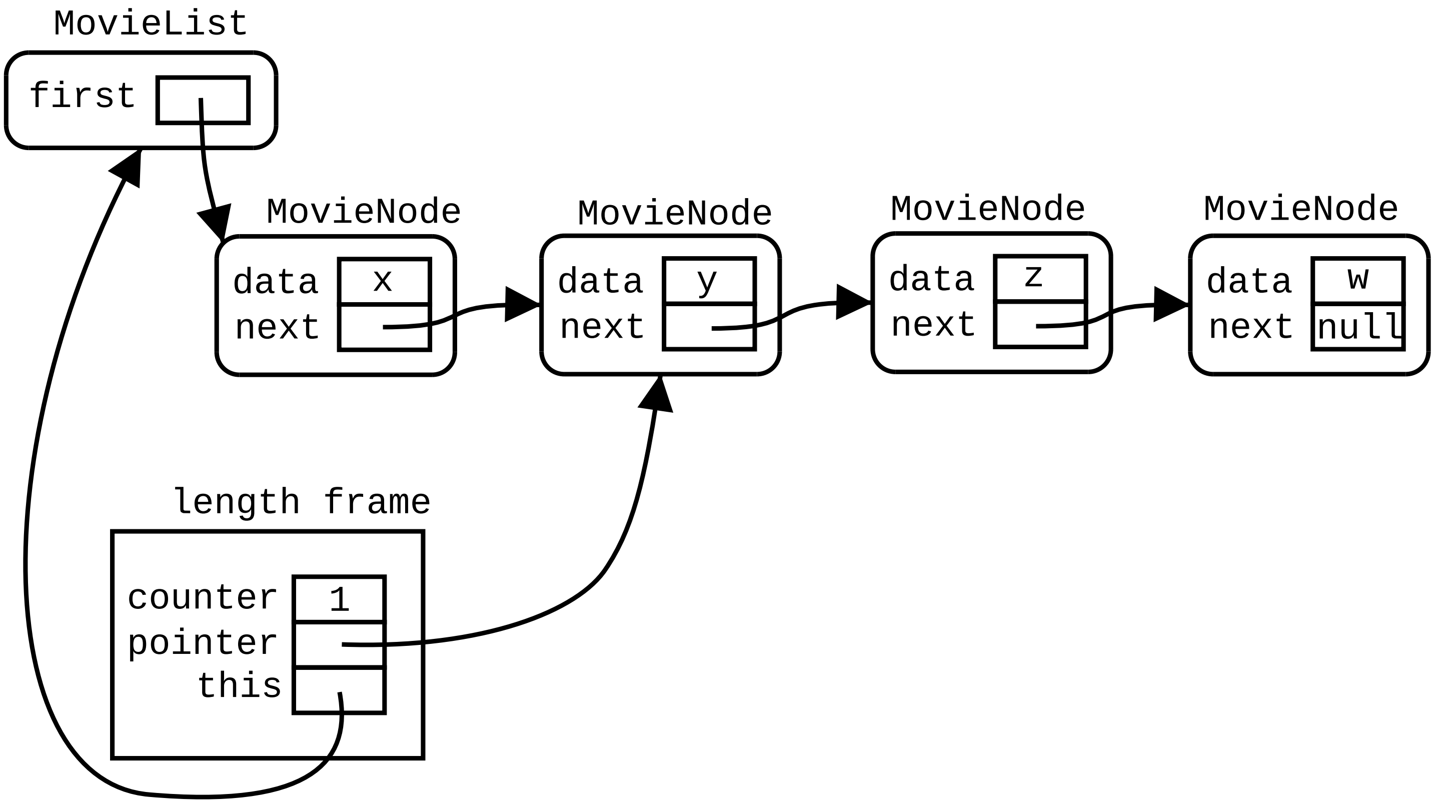
Linked Lists

```
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();
  }
}
```

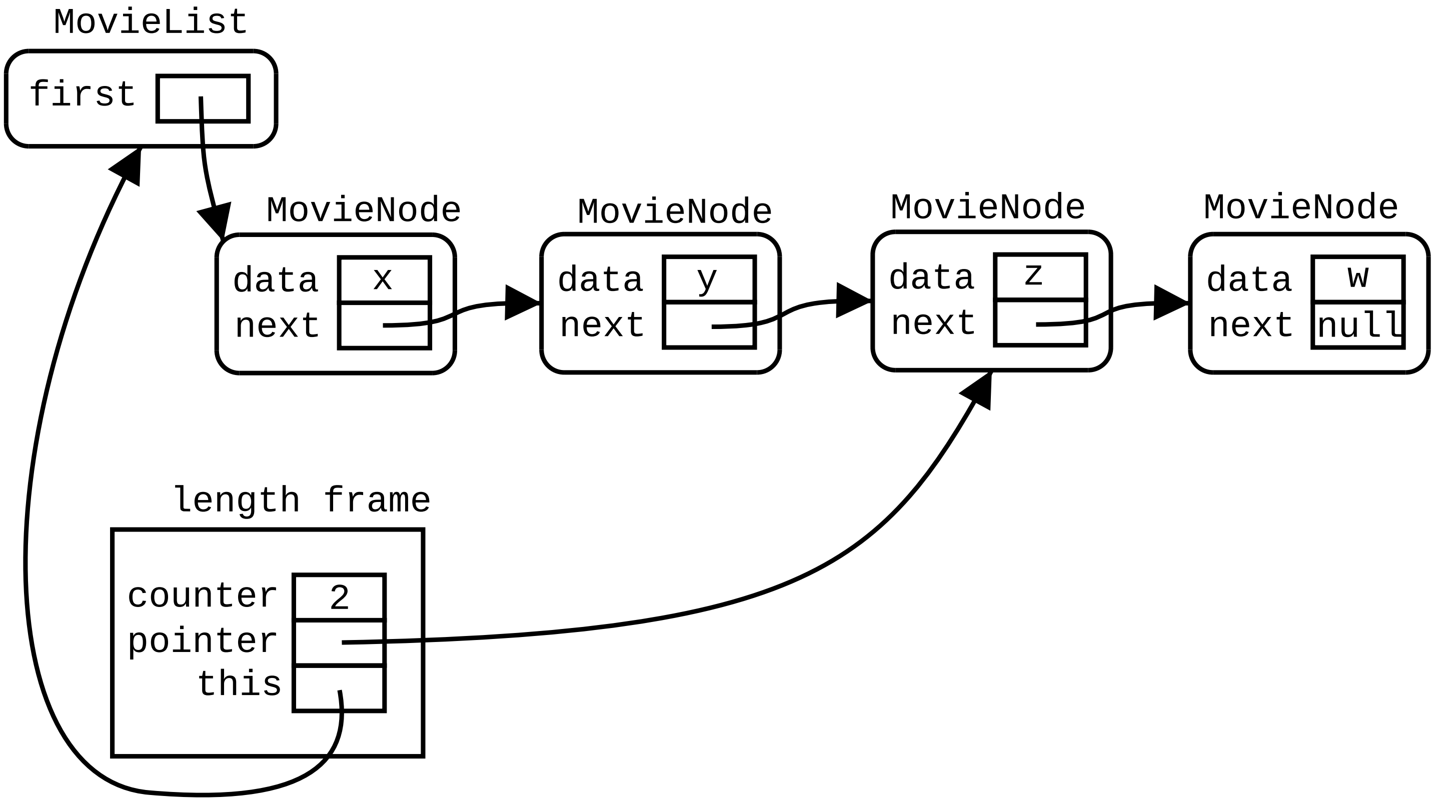
Linked Lists



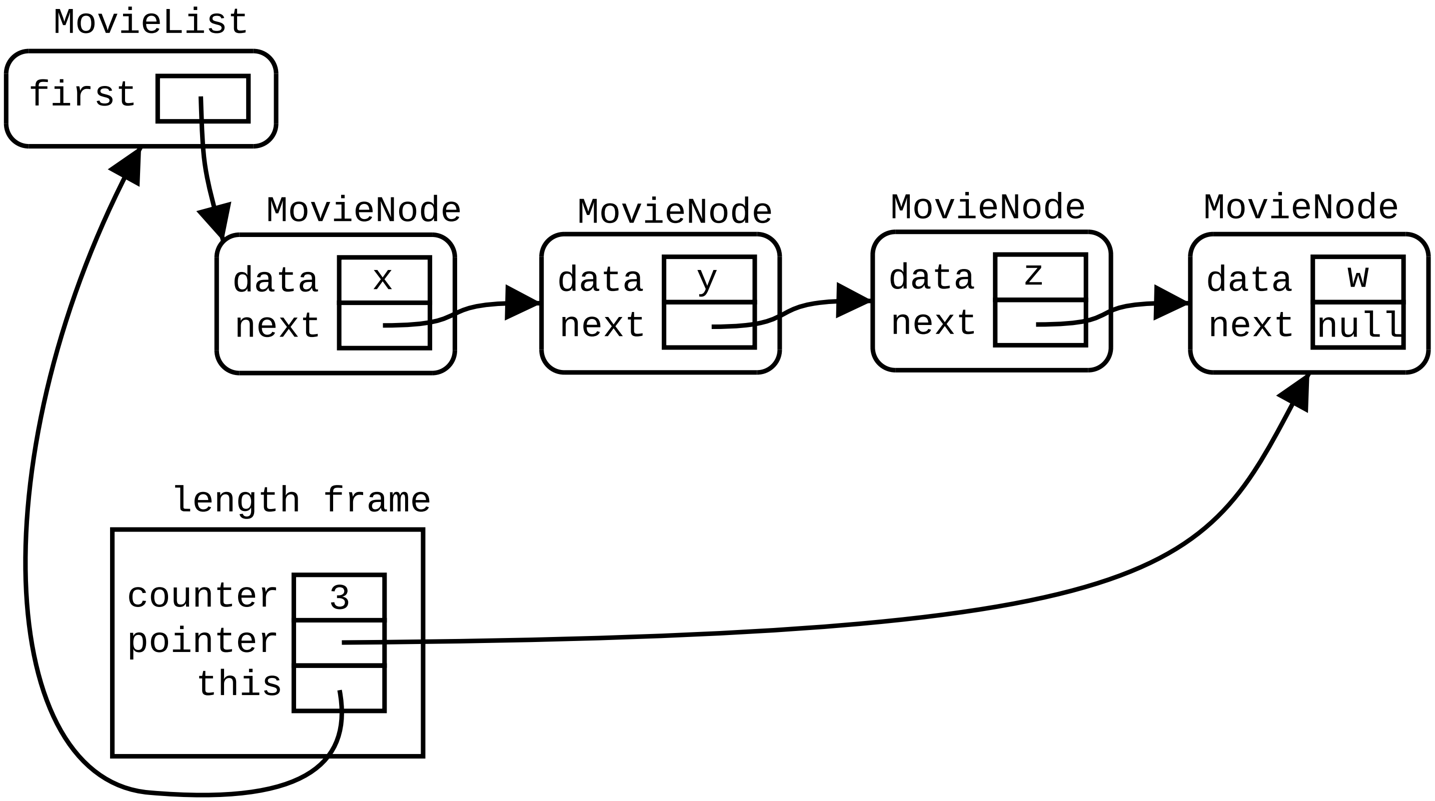
Linked Lists



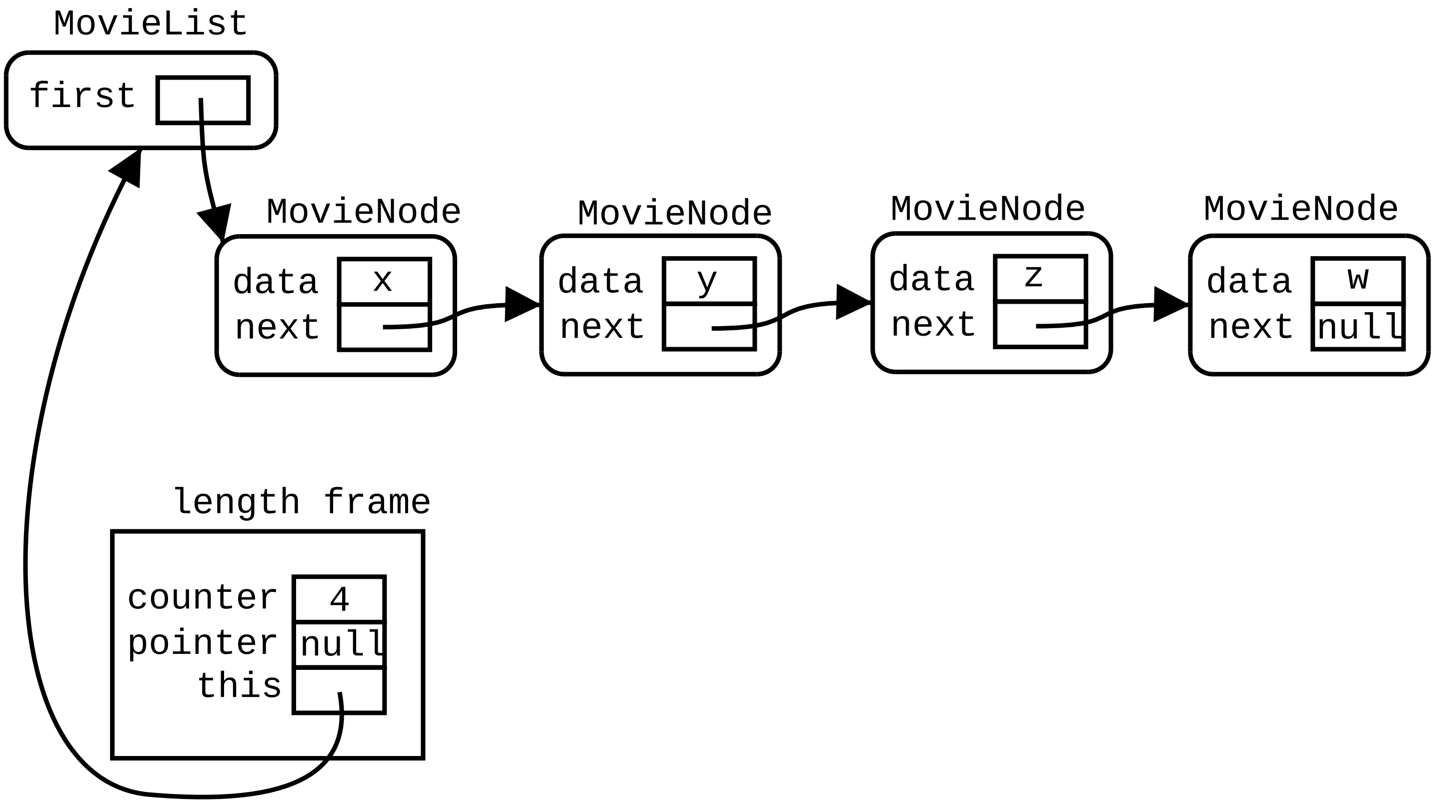
Linked Lists



Linked Lists



Linked Lists



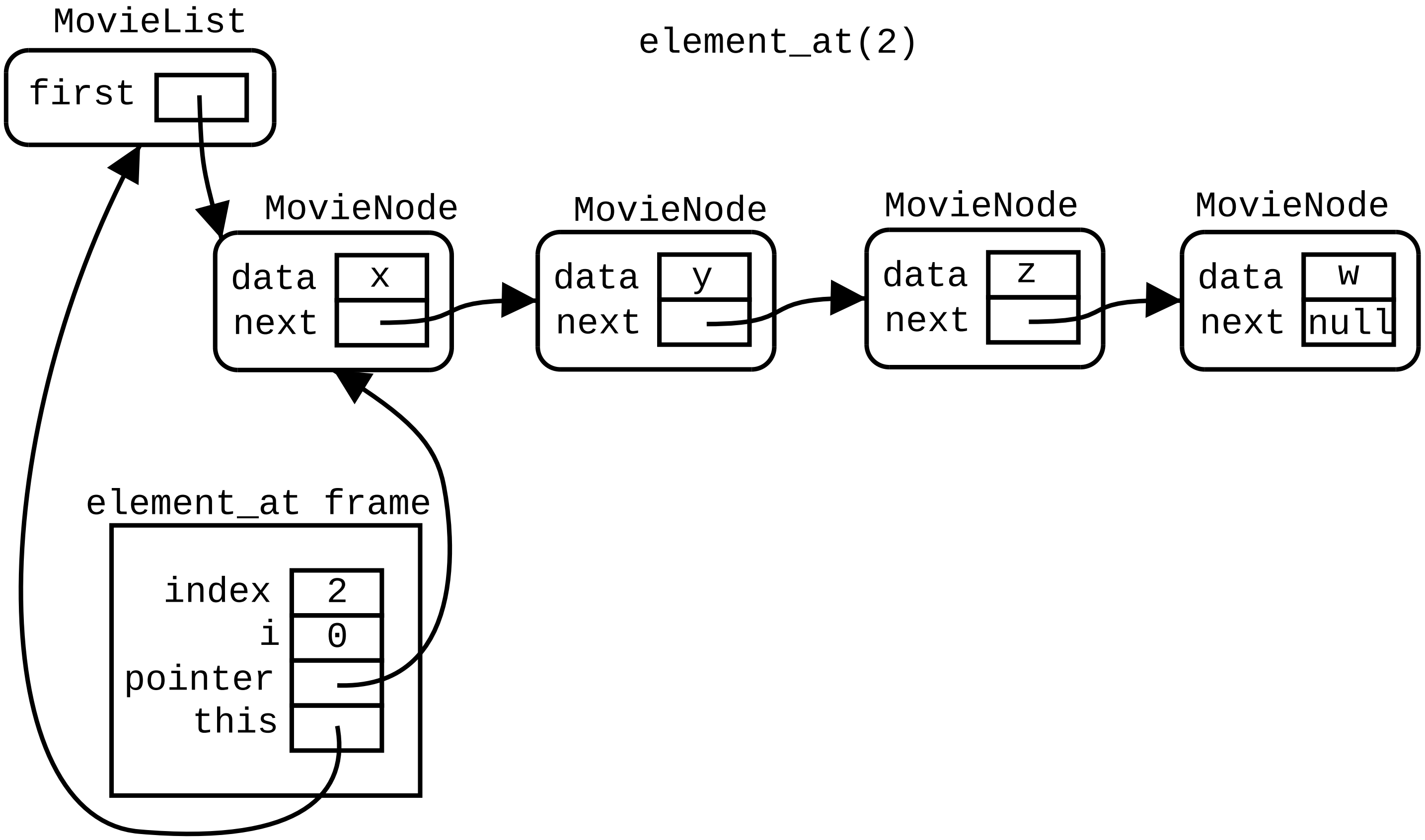
Linked Lists

```
class MovieList {
  private MovieNode first;
  //...
  public Movie element_at(int index)
  throws IndexOutOfBoundsException
  {
    if (index < 0)
      throw new IndexOutOfBoundsException();
    int i = 0;
    MovieNode pointer = first;
    while (pointer != null && i < index) {
      pointer = pointer.get_next();
      i++;
    }
    if (pointer == null)
      throw new IndexOutOfBoundsException();
    return pointer.get_movie();
  }
}
```

Linked Lists

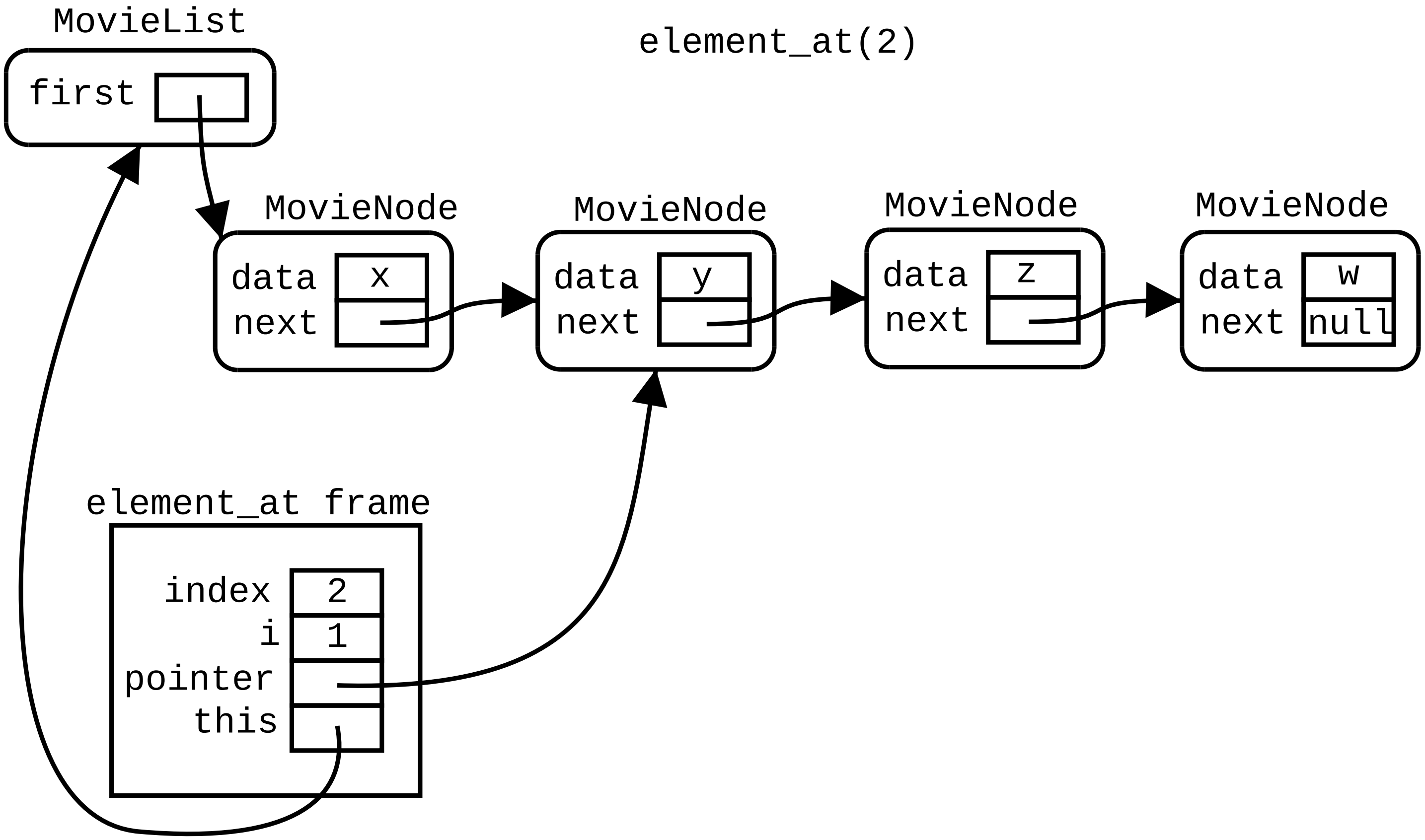
```
class Test {
  public static void main(String[] args)
  {
    MovieList l = new MovieList();
    Movie w = new Movie(“abc”,“def”);
    Movie x = new Movie(“bca”,“efd”);
    Movie z = new Movie(“cba”,“fef”);
    Movie y = new Movie(“xxx”,“yyy”);
    l.add(w);
    l.add(z);
    l.add(y);
    l.add(x);
    Movie m = l.element_at(2);
  }
}
```

Linked Lists

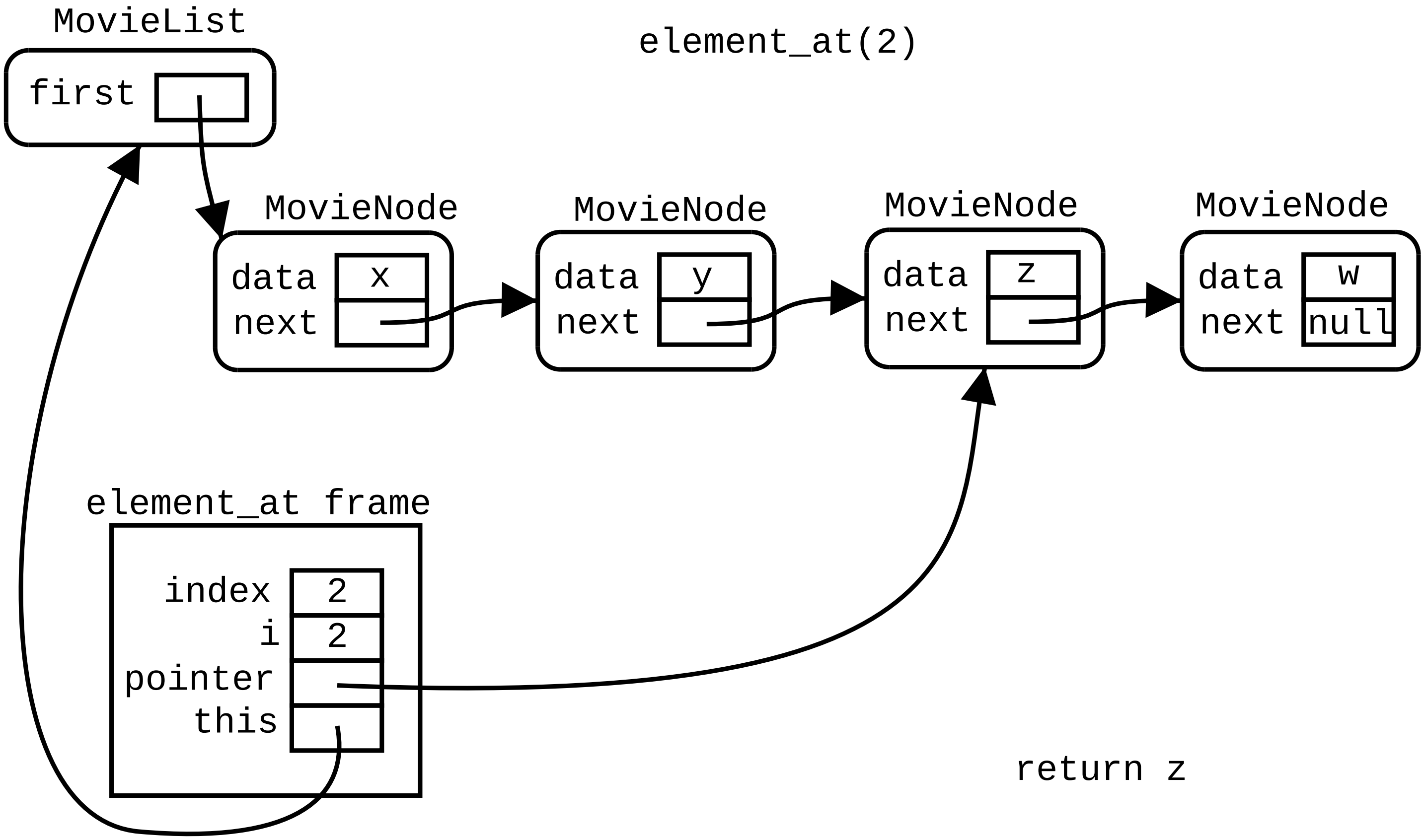


Linked Lists

element_at(2)



Linked Lists



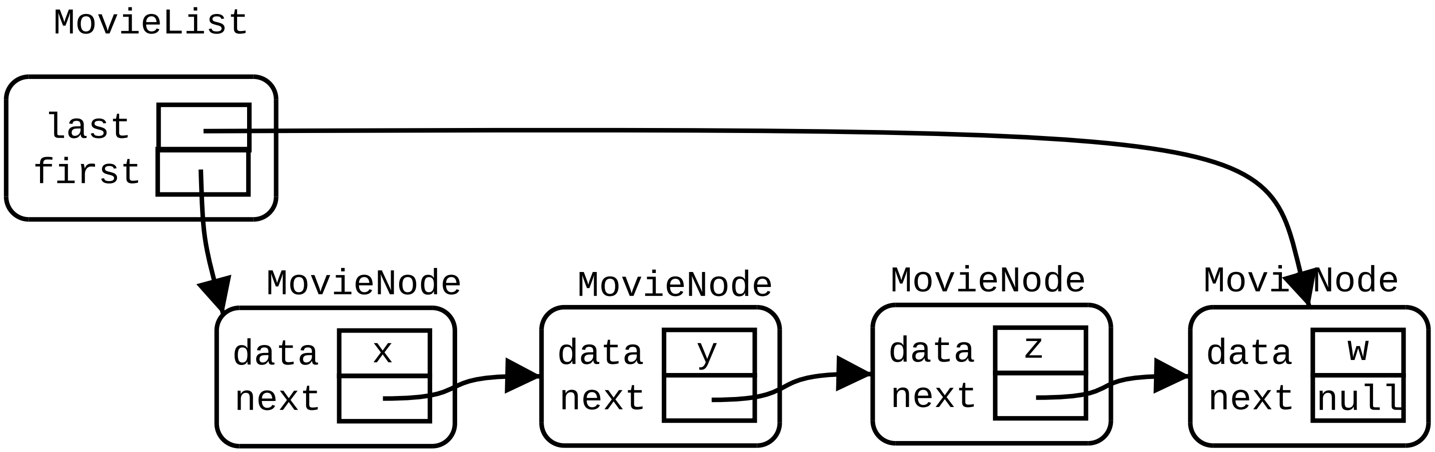
Linked Lists

```
class MovieList {
  private MovieNode first;
  //...
  public void add_at_end(Movie m)
  {
    MovieNode new_node = new MovieNode(m, null);
    MovieNode pointer;
    if (first == null) {
      first = new_node;
    }
    else {
      pointer = first;
      while (pointer.get_next() != null) {
        pointer = pointer.get_next();
      }
      pointer.set_next(new_node);
    }
  }
}
```

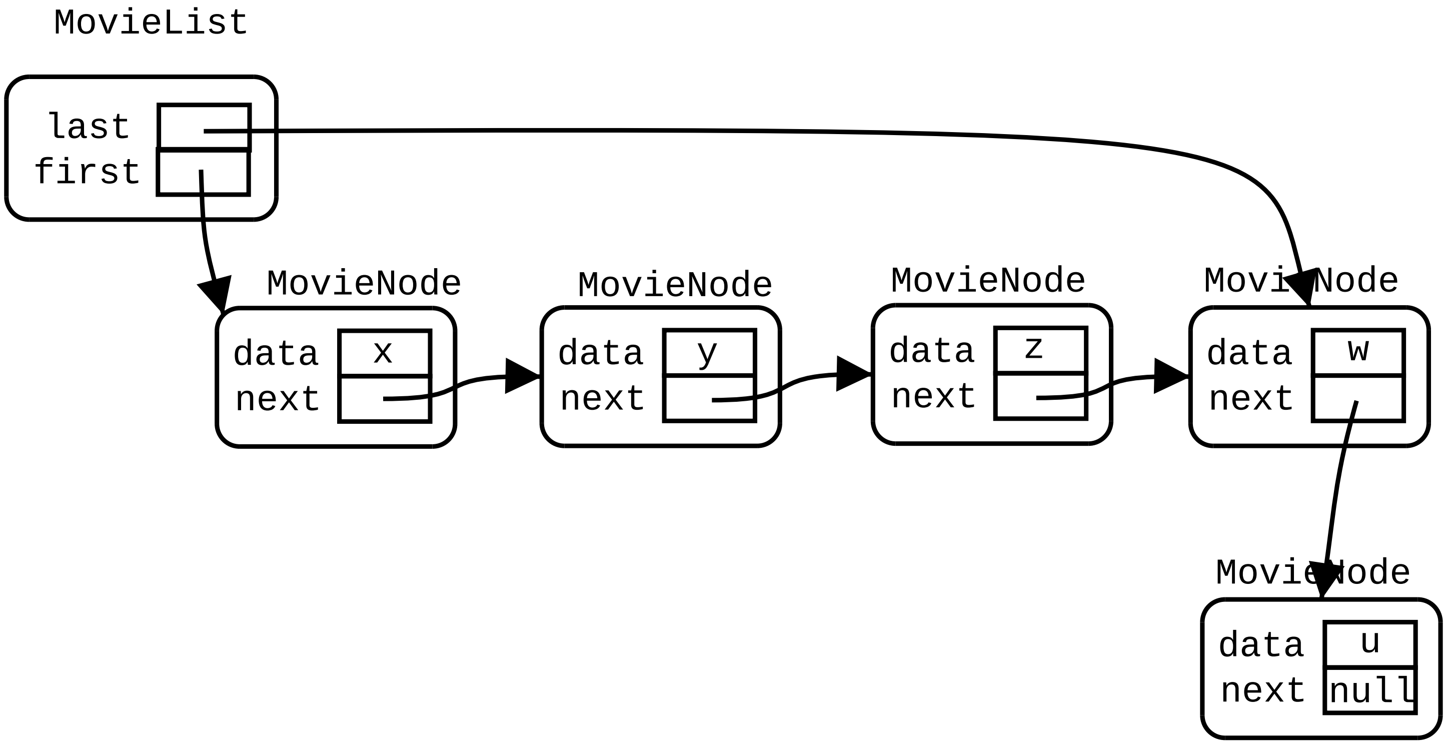
Linked Lists

```
class MovieList {
  private MovieNode first, last;
  //...
  public void add_at_end(Movie m)
  {
    MovieNode new_node = new MovieNode(m, null);
    if (first == null) {
      first = new_node;
      last = new_node;
    }
    else {
      last.set_next(new_node);
      last = new_node;
    }
  }
}
```

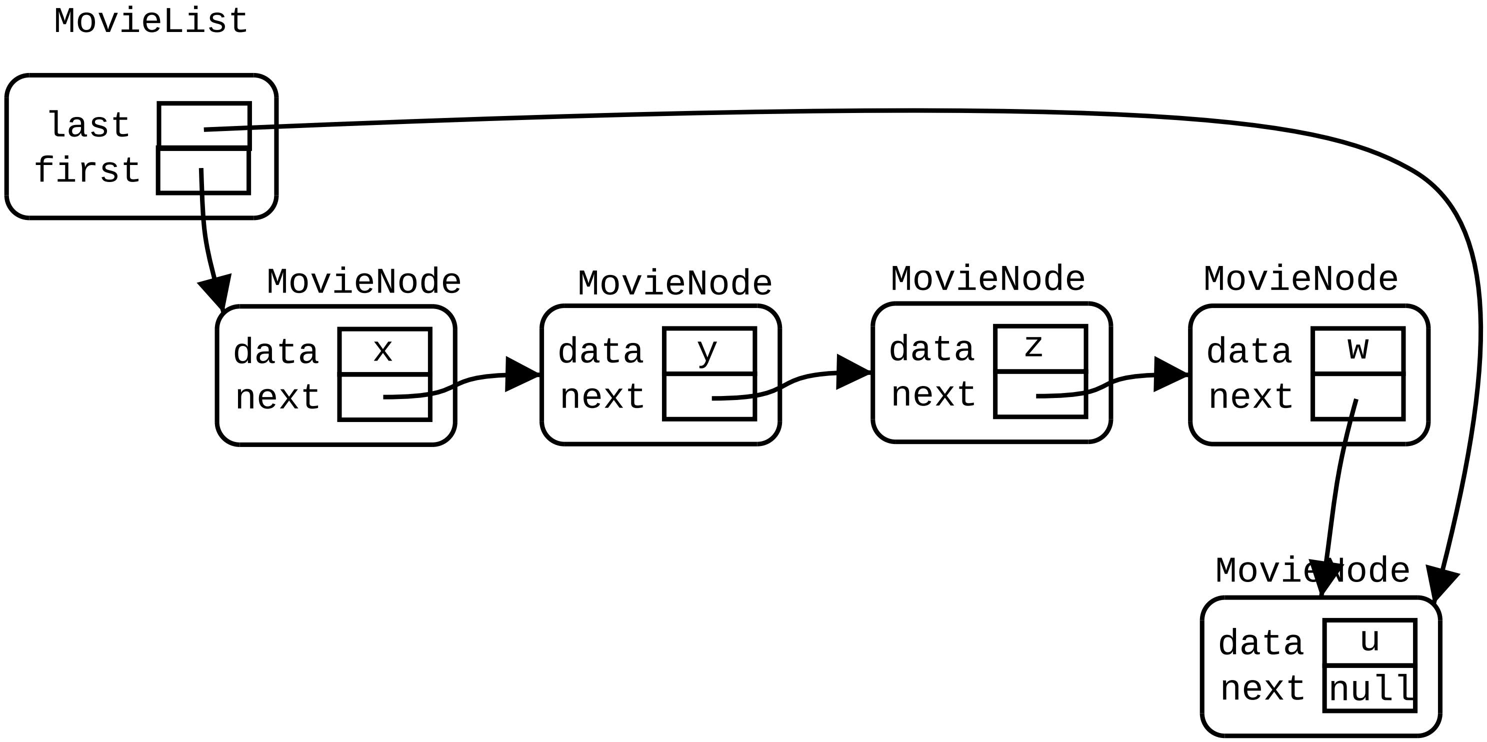
Linked-lists



Linked-lists



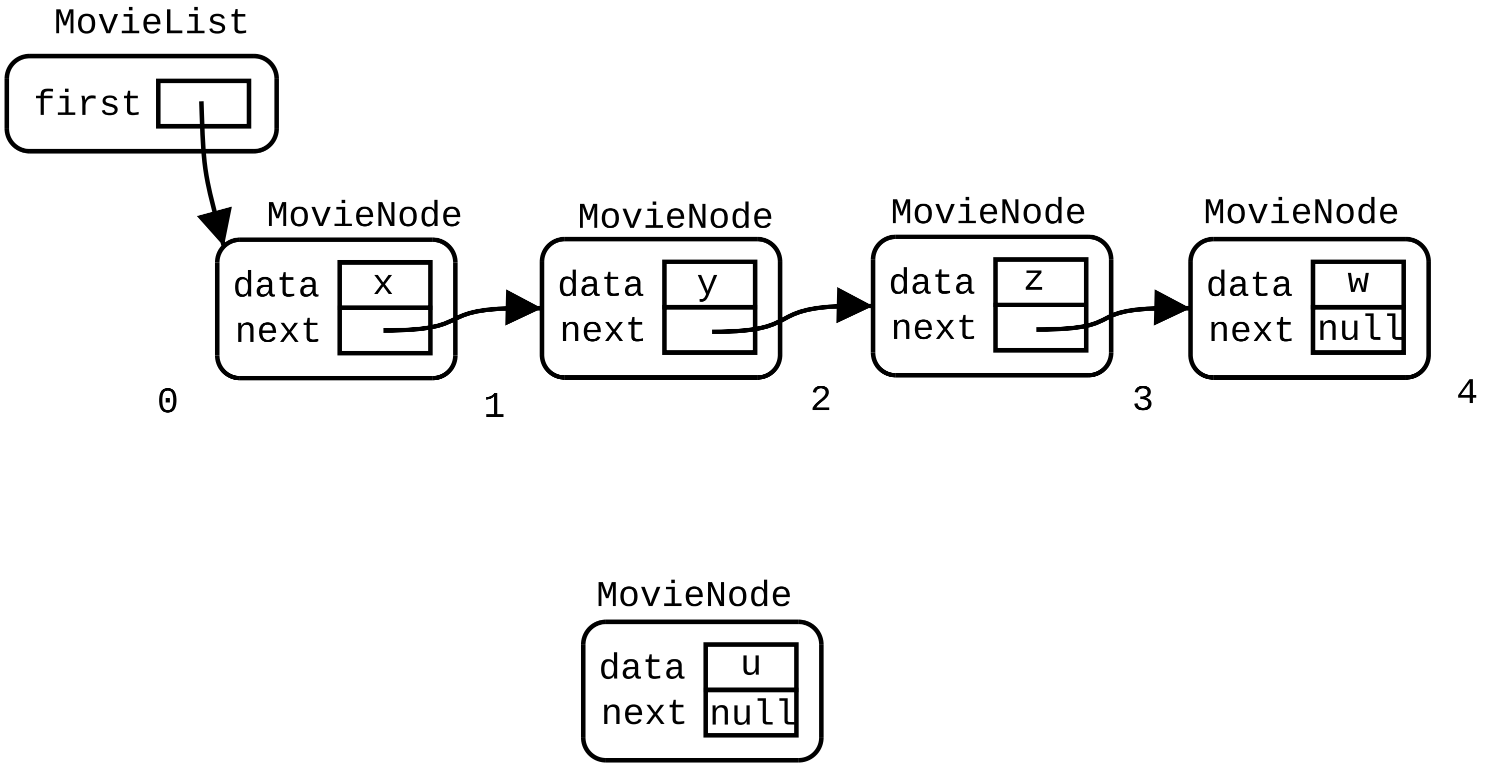
Linked-lists



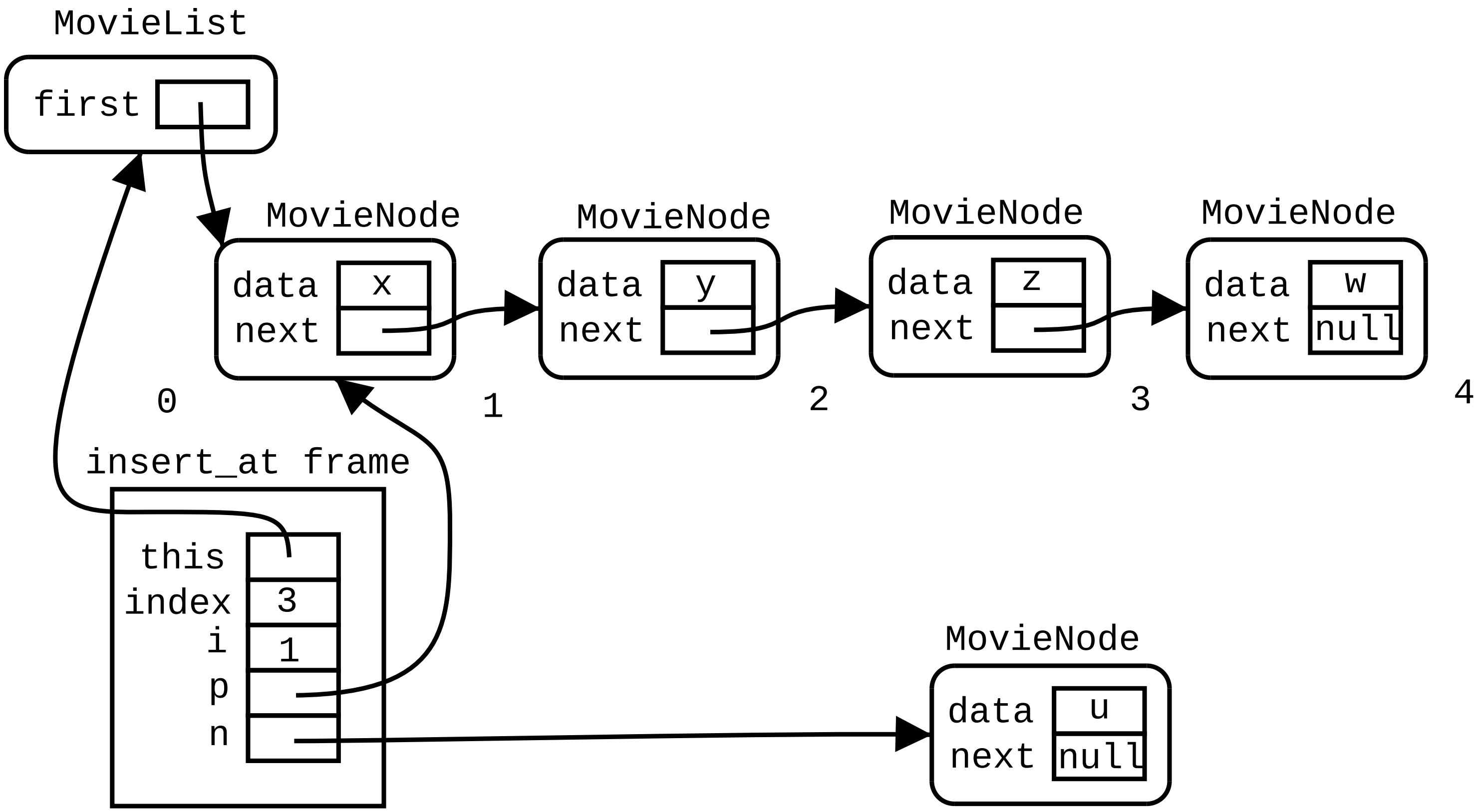
Linked-lists

```
public void insert_at(Movie m, int index)
throws IndexOutOfBoundsException {
  if (index < 0)
    throw new IndexOutOfBoundsException();
  MovieNode n = new MovieNode(m, null);
  if (index == 0) {
    n.set_next(first);
    first = n;
  }
  else {
    MovieNode p = first;
    int i = 1;
    while (i < index && p != null) {
      p = p.get_next();
      i++;
    }
    if (p == null)
      throw new IndexOutOfBoundsException();
    n.set_next(p.get_next());
    p.set_next(n);
  }
}
```

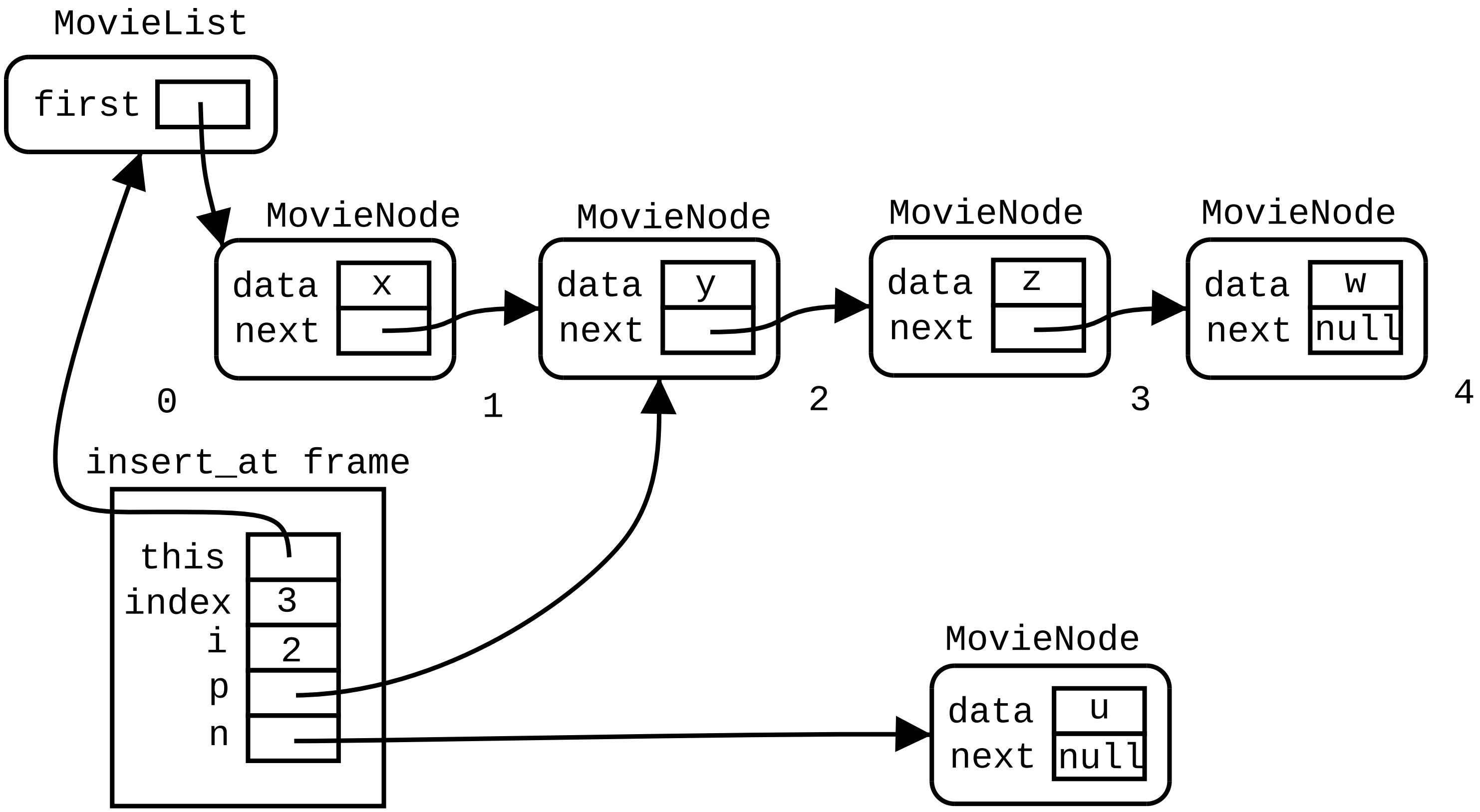
Linked-lists



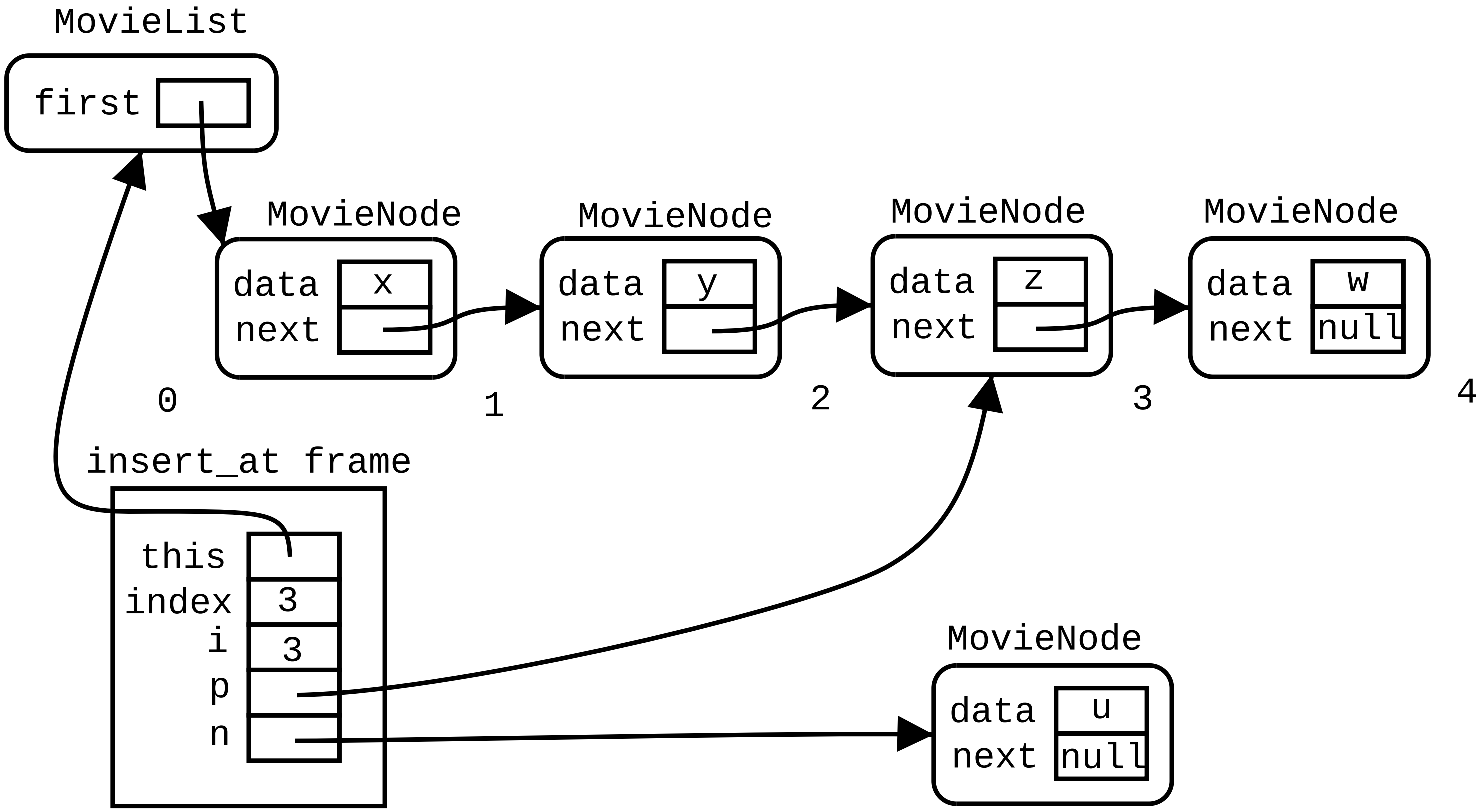
Linked-lists



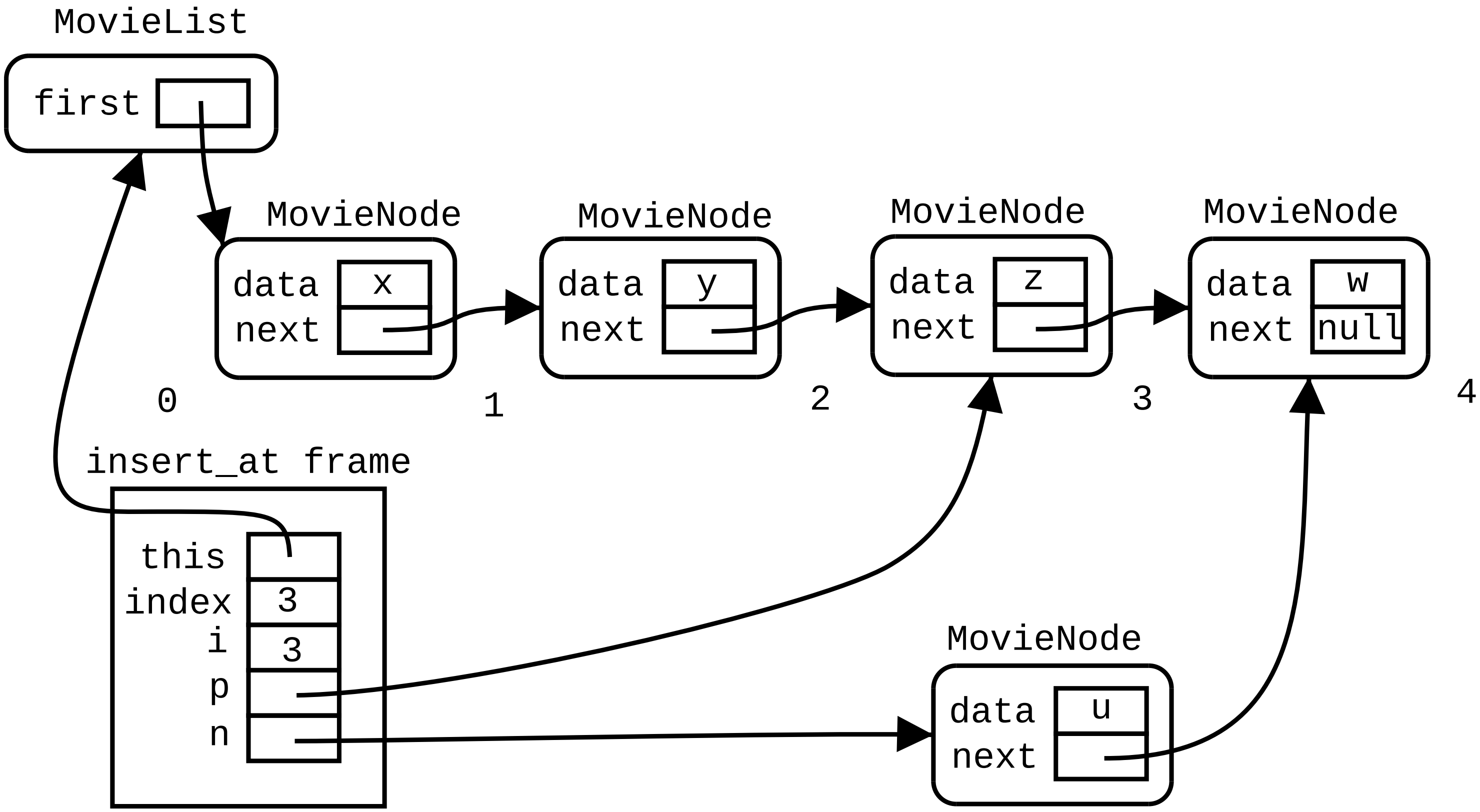
Linked-lists



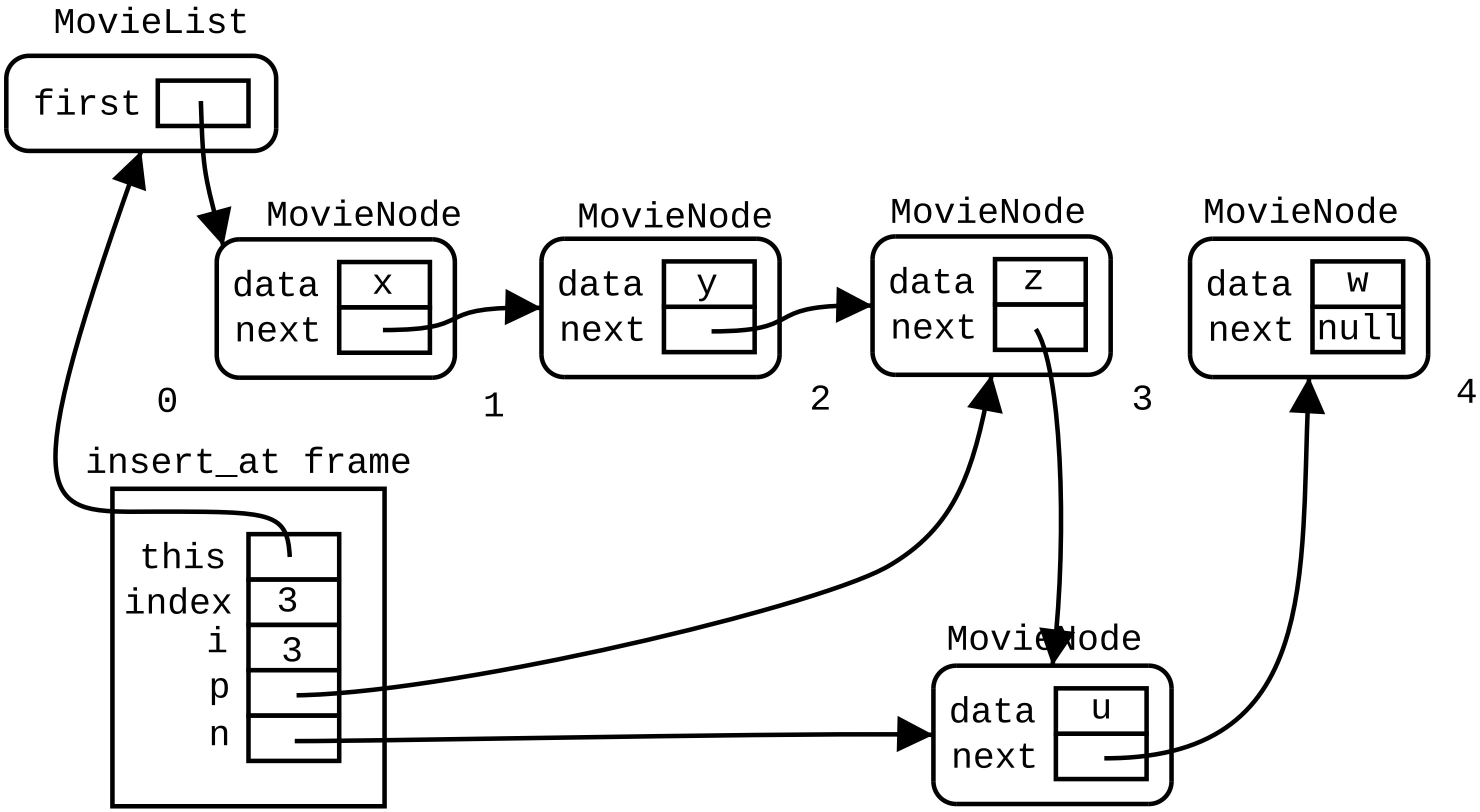
Linked-lists



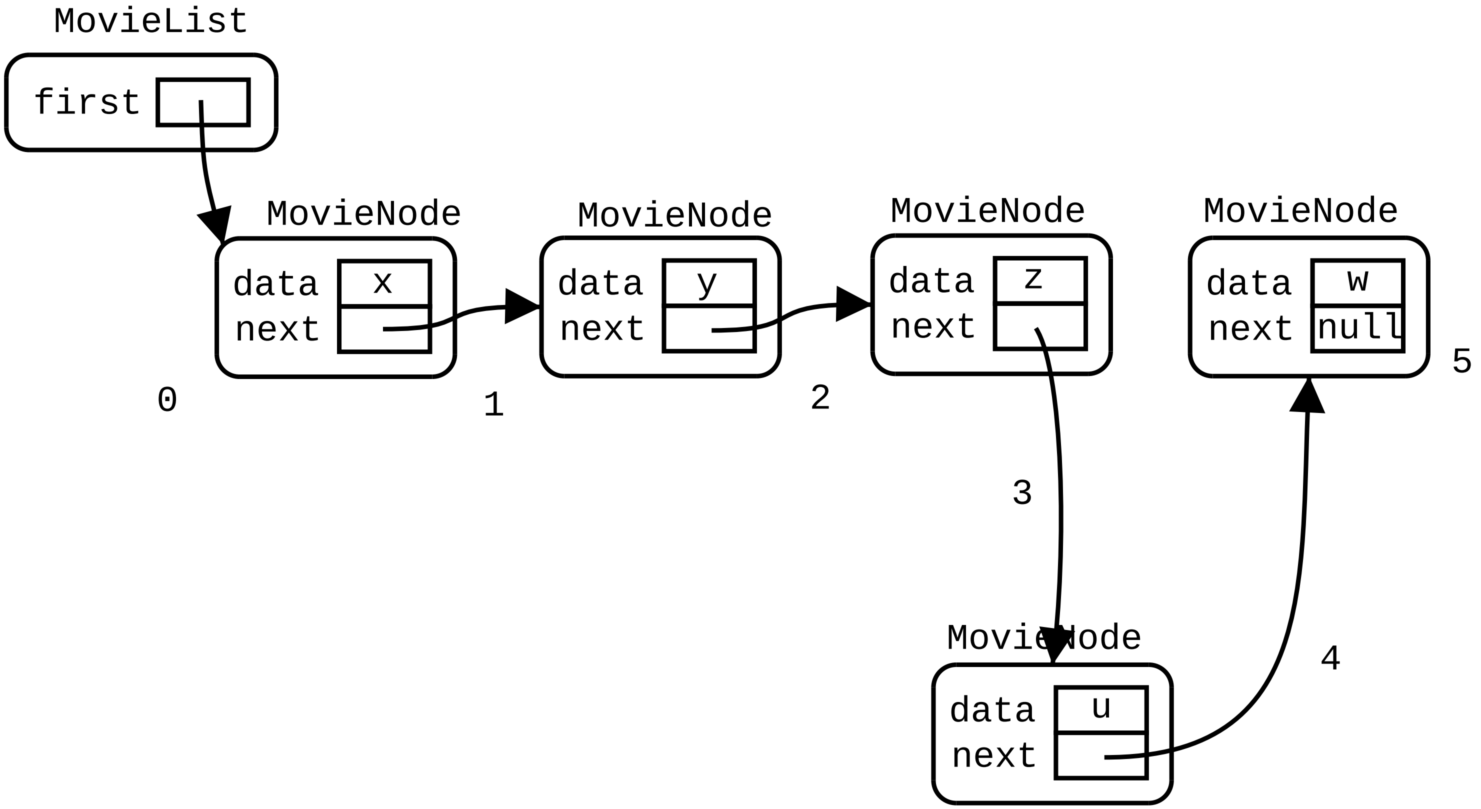
Linked-lists



Linked-lists



Linked-lists



Linked-lists

```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  public void add(Movie m)
  throws IndexOutOfBoundsException
  {
    insert_at(m, 0);
  }
  public void add_at_end(Movie m)
  throws IndexOutOfBoundsException
  {
    insert_at(m, length());
  }
}
```

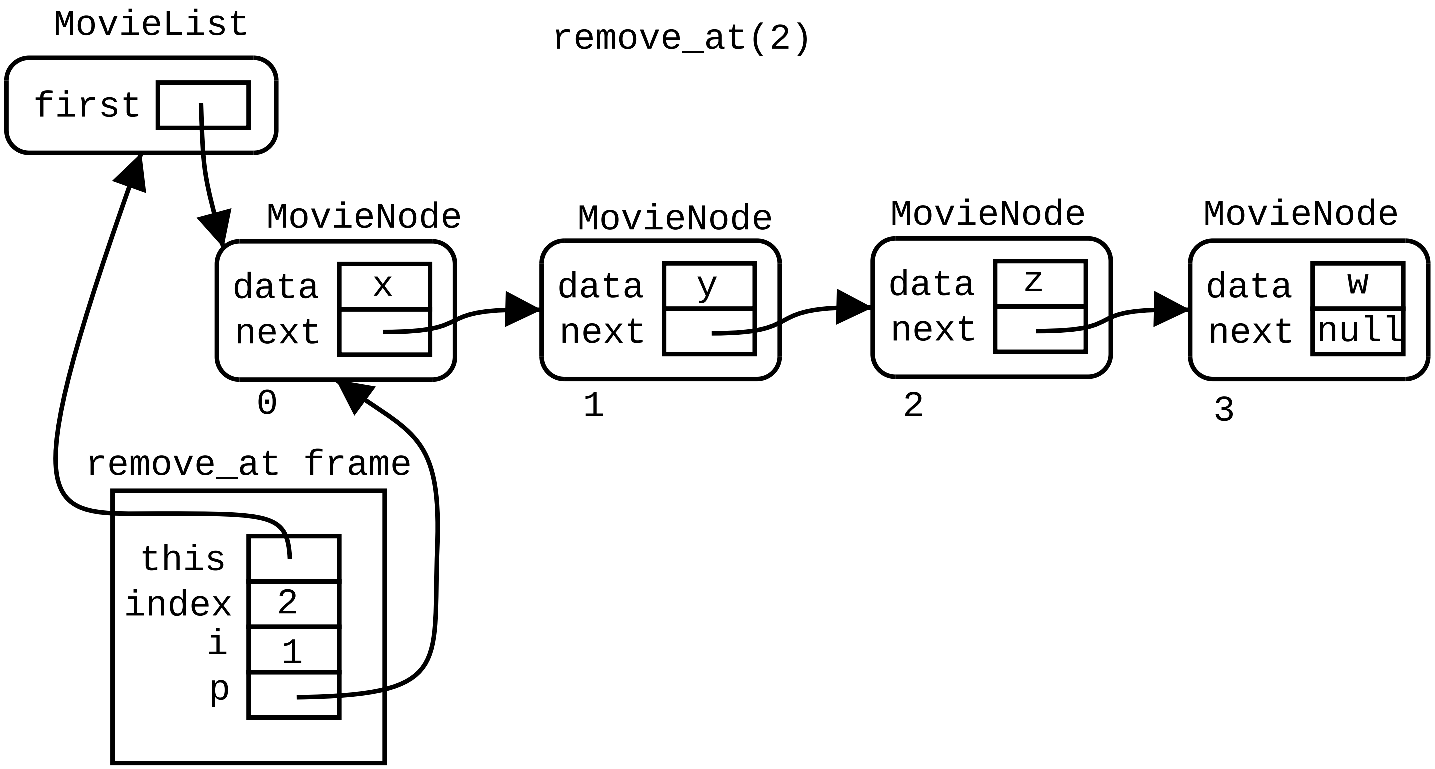
Linked-lists

```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  public void remove_first()
  throws IndexOutOfBoundsException
  {
    if (first == null)
      throw new IndexOutOfBoundsException();
    first = first.get_next();
  }
}
```

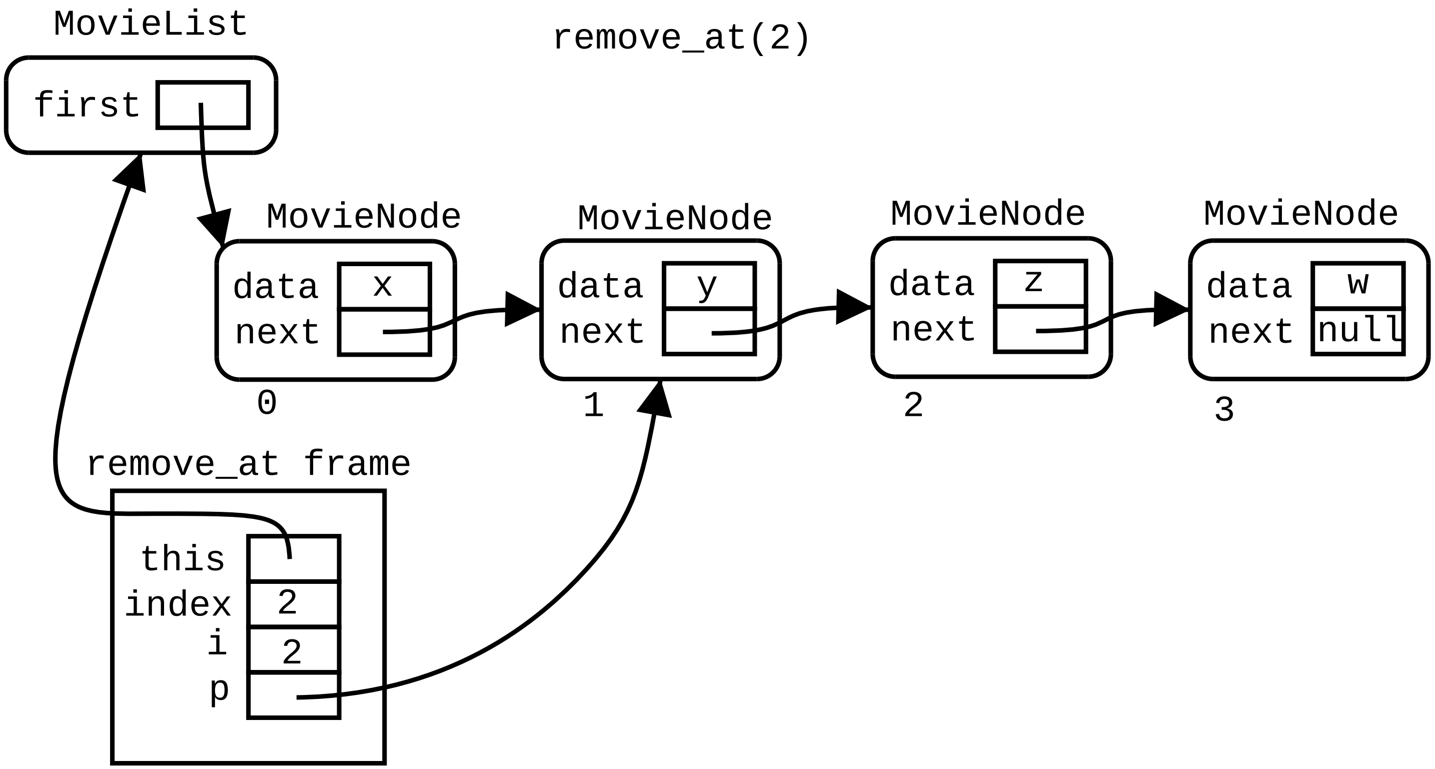
Linked-lists

```
public void remove_at(int index)
throws IndexOutOfBoundsException
{
  if (index < 0)
    throw new IndexOutOfBoundsException();
  if (index == 0) {
    first = first.get_next();
  }
  else {
    MovieNode p = first;
    int i = 1;
    while (i < index && p.get_next() != null) {
      p = p.get_next();
      i++;
    }
    if (p.get_next() == null)
      throw new IndexOutOfBoundsException();
    p.set_next(p.get_next().get_next());
  }
}
```

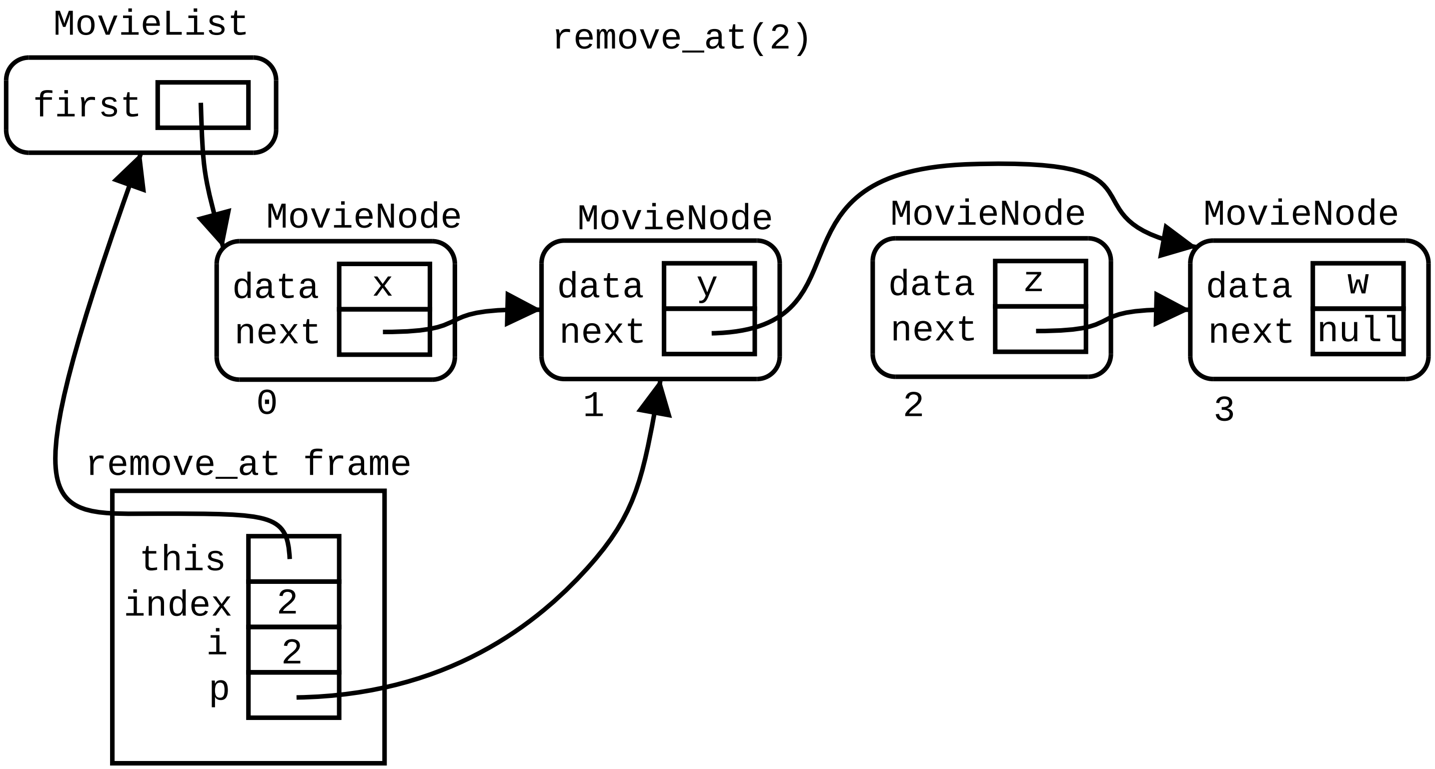
Linked-lists



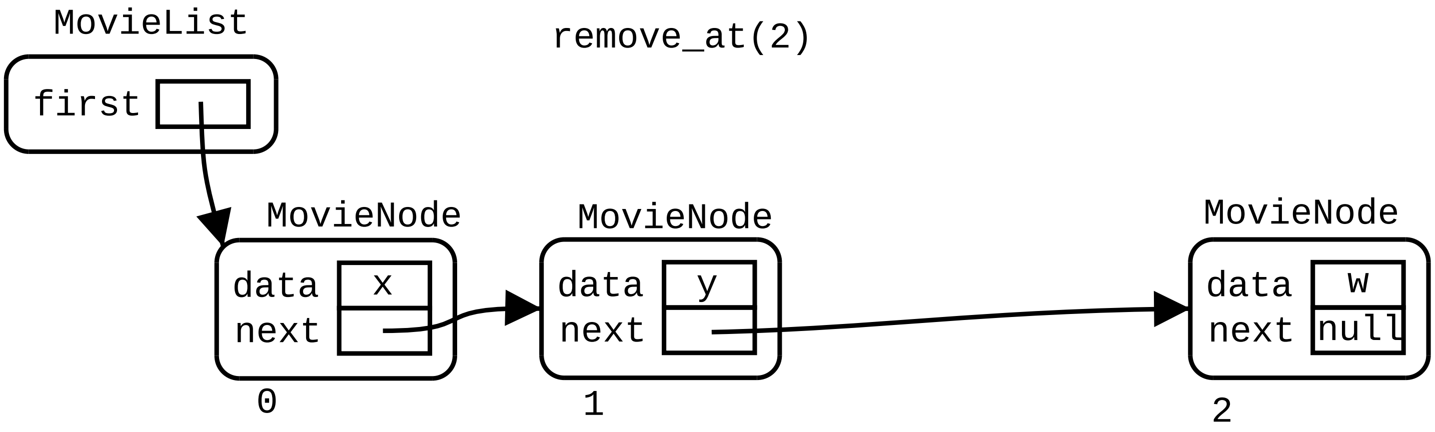
Linked-lists



Linked-lists



Linked-lists



Linked-lists

```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  boolean equals(MovieList l)
  {
    if (l == null) return false;
    if (first == null) return l.first == null;
    return first.equals(l.first);
  }
}
```

Linked-lists

```
class Movie {
  // ...
  public boolean equals(Movie m) { ... }
}
class MovieNode {
  Movie data;
  MovieNode next;
  //...
  public boolean equals(MovieNode n) {
    if (n == null) return false;
    boolean equal_data = data.equals(n.data);
    if (next == null && n.next == null)
      return equal_data;
    return equal_data && next.equals(n.next);
  }
}
```

Queues

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

Queues

```
public class Node
{
  private Object data;
  private Node next;

  public Node(Object d, Node n)
  {
    data = d;
    next = n;
  }
  public Object get_data() { return data; }
  public Node get_next() { return next; }
  public void set_data(Object d) {
    data = d;
  }
  public void set_next(Node n) {
    next = n;
  }
}
```

Queues

```
public class LinkedListQueue implements Queue
{
  private Node first, last;

  public LinkedListQueue()
  {
    first = null;
    last  = null;
  }

  public boolean isEmpty()
  {
    return (first == null);
  }

  public Object peek()
  {
    if (!isEmpty()) return first.get_data();
    return null;
  }
  public void enqueue(Object obj)
  {
    Node nn = new Node(obj, null);
    if (isEmpty()) {
      first = nn;
      last  = nn;
    }
    else {
      last.set_next(nn);
      last = nn;
    }
  }

  public void dequeue()
  {
    if (!isEmpty()) {
      first = first.get_next();
    }
  }
}
```

The end