Announcement

- No class Friday April 9th.
- Last day of lectures: Tuesday, April 13 at the same time and place.
- Tutorials TBA

Exception handling

- Some exceptions arise without an explicit throw.
- Some standard exceptions

```
Exception
  RunTimeException
    IndexOutOfBounds
    StringIndexOutOfBounds
    ArithmeticException (e.g. division by 0)
    NullPointerException
```

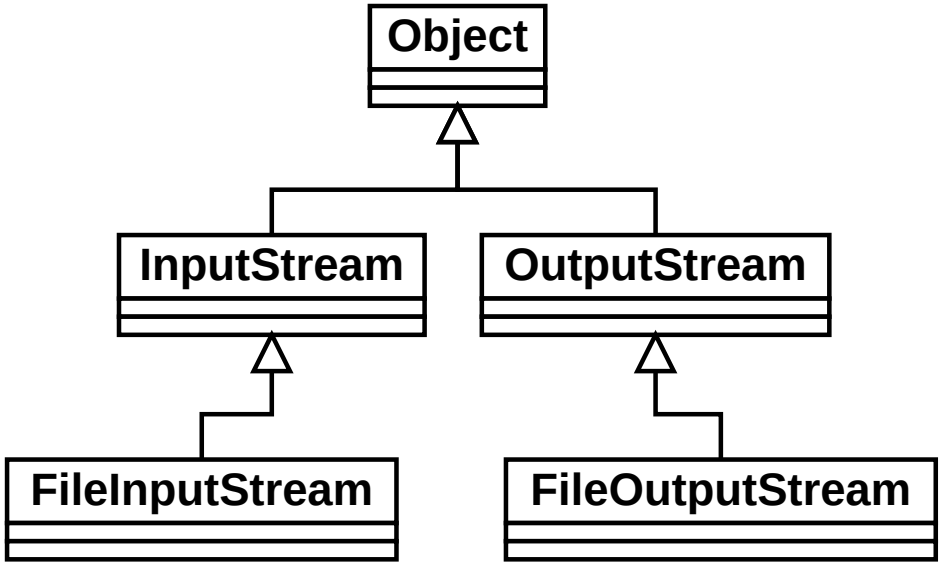
IO

- When an object is created, it is destroyed whenever there are no more references to it.
- Persistence: saving the state of an object so that it outlives the execution of a program
- Input-Output Operations:
  - Saving information
  - Loading information
- Files
- Streams

I/O

- `java.io` package
- Streams
  - A *stream* is a sequence of elements. Possibly with no fixed size.
  - I/O Streams (char vs. byte streams.)
  - Stream operations: read from the stream, and write to the stream.
  - Associate an I/O stream with a file.

I/O


Object
InputStream
OutputStream
FileInputStream
FileOutputStream

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.

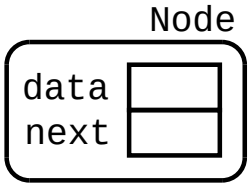
The List ADT

- 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;
}
```



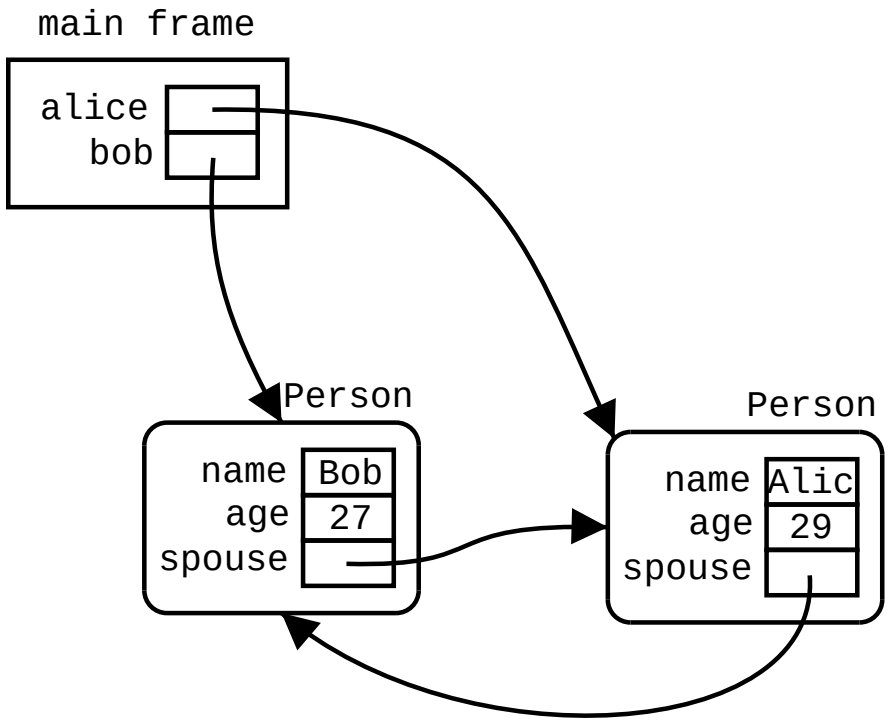
- A recursive data-structure has references to objects of its own type

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;
  }
}
```

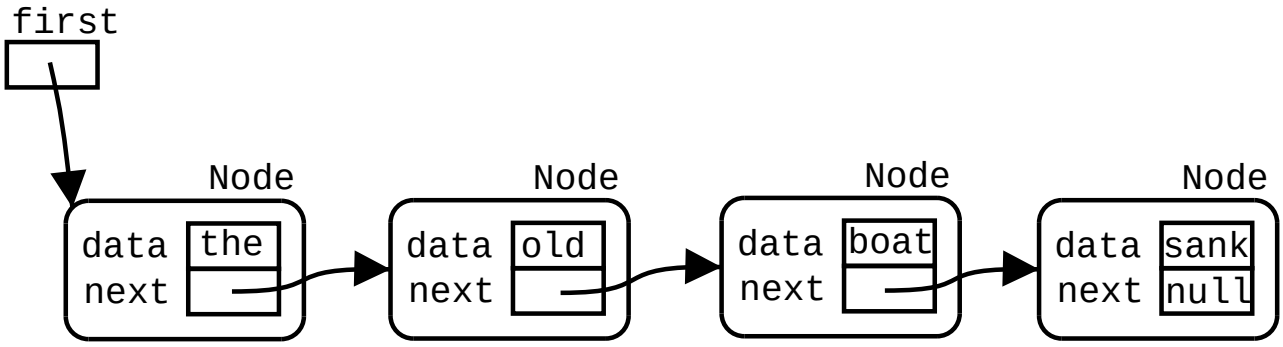
Recursive data-structures

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



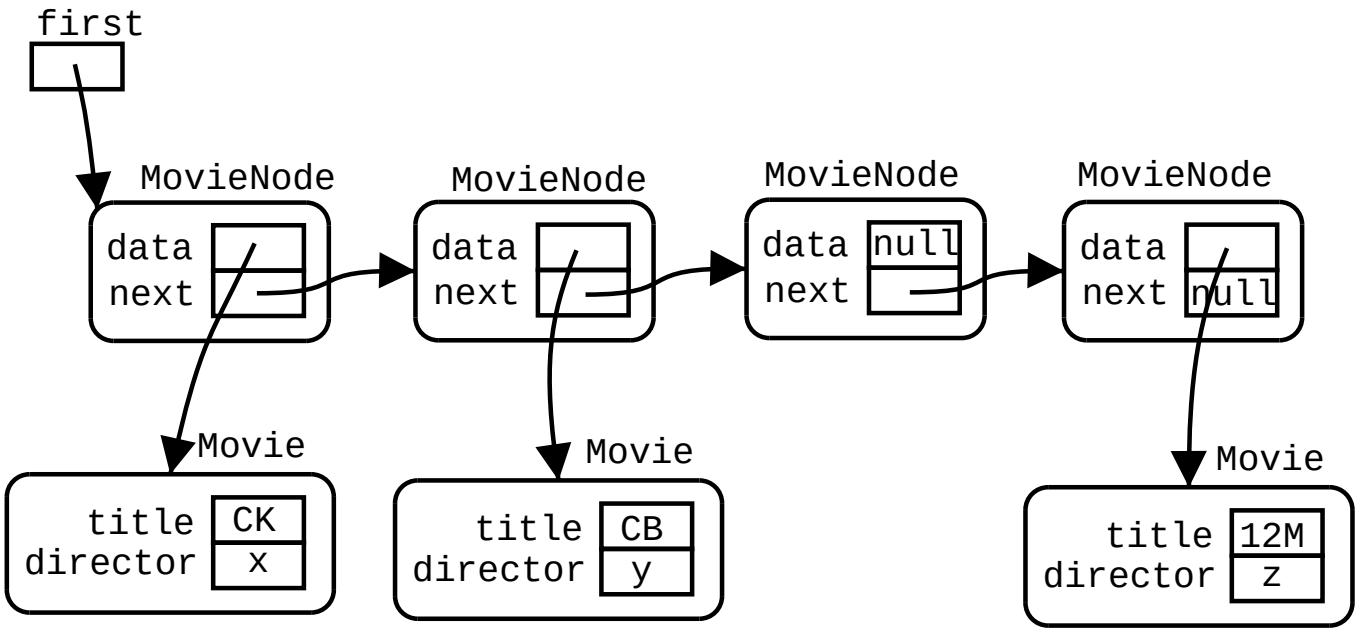
Linked Lists

```
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

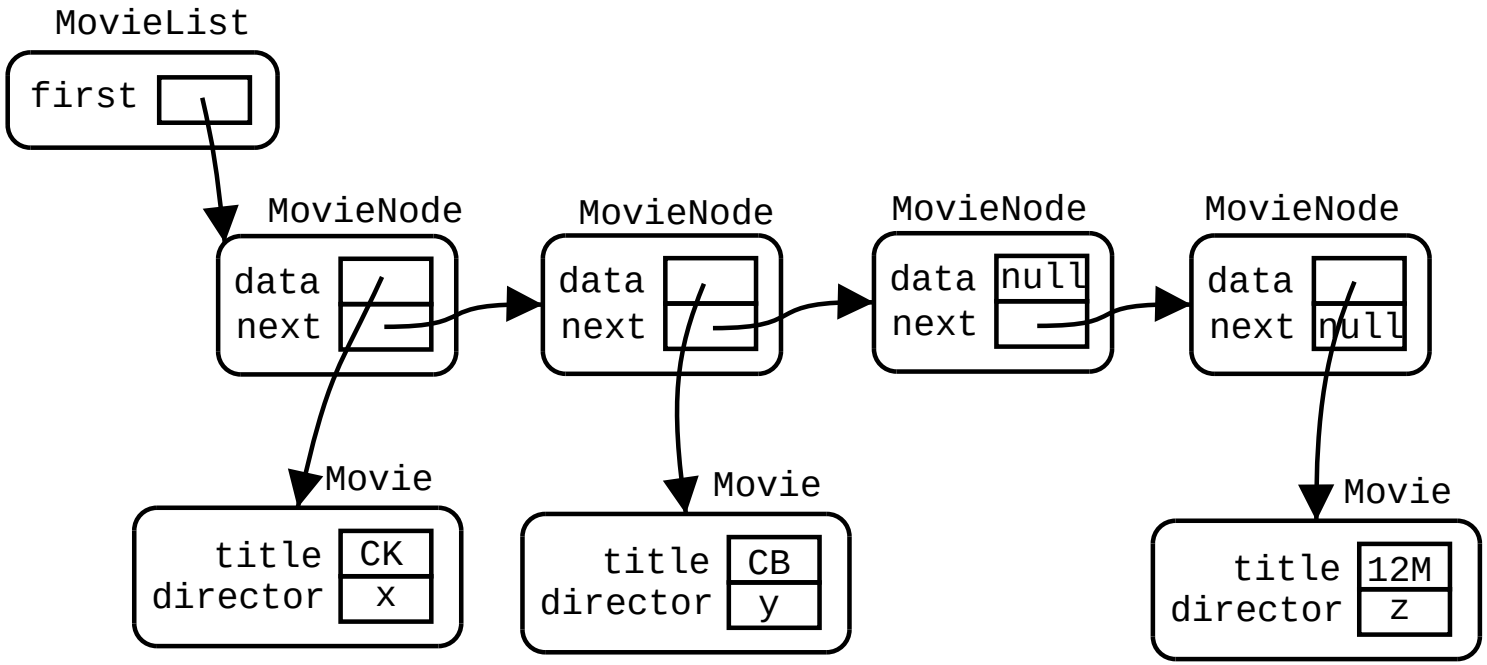
```
class Movie {
  String title, director;
  // ...
}

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



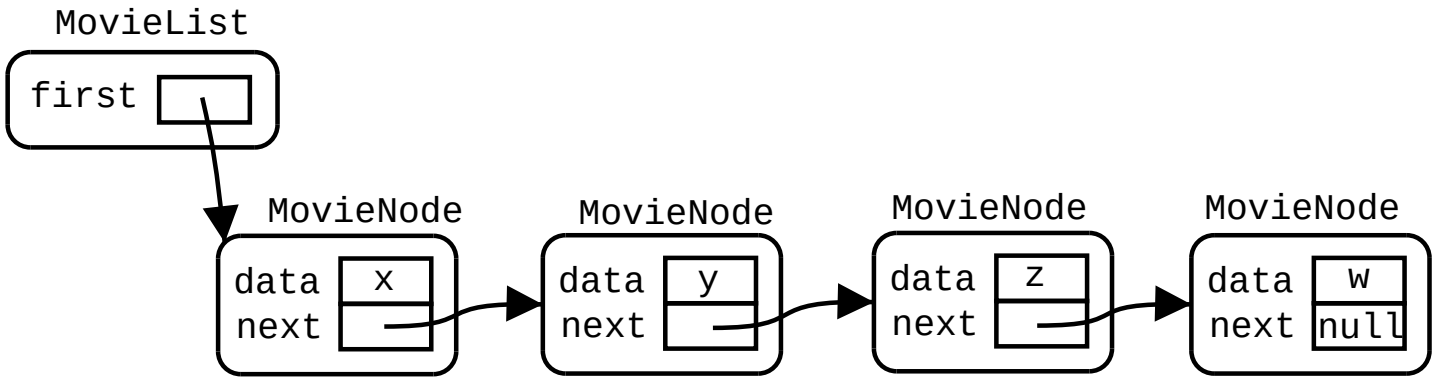
Linked Lists

```
class MovieList {
  private MovieNode first;

  public MovieList() { first = null; }

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

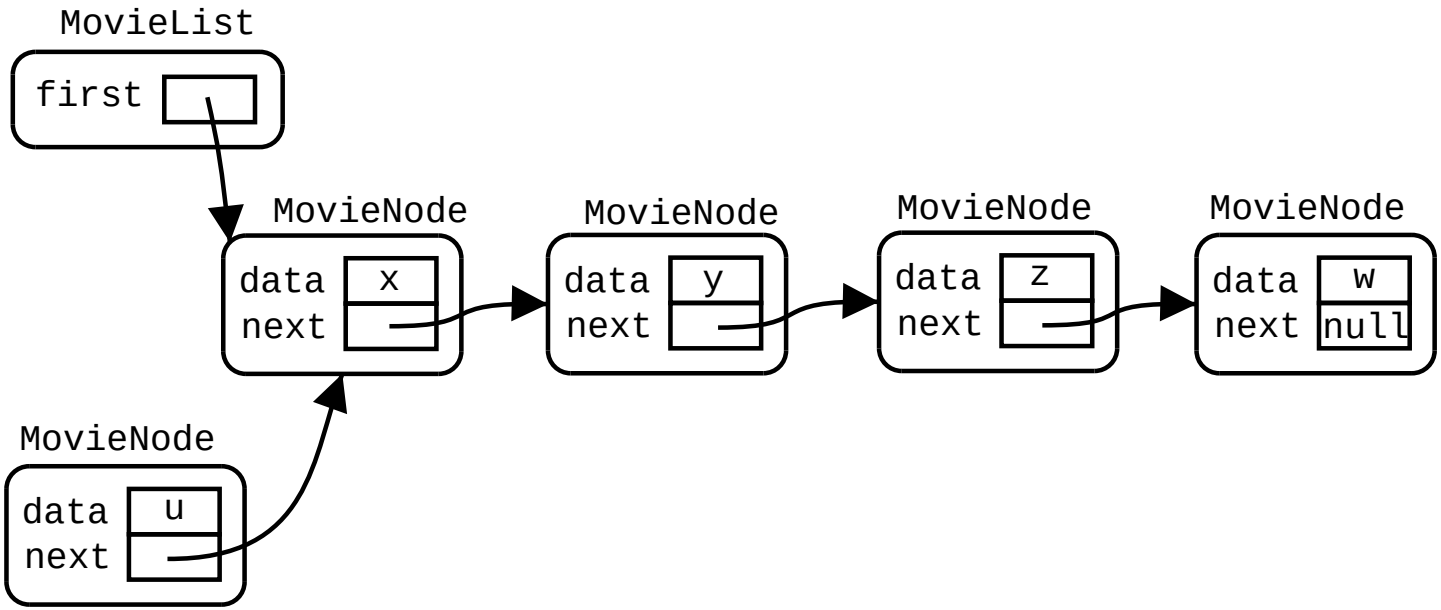
Linked Lists

```
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);
  }
}
```

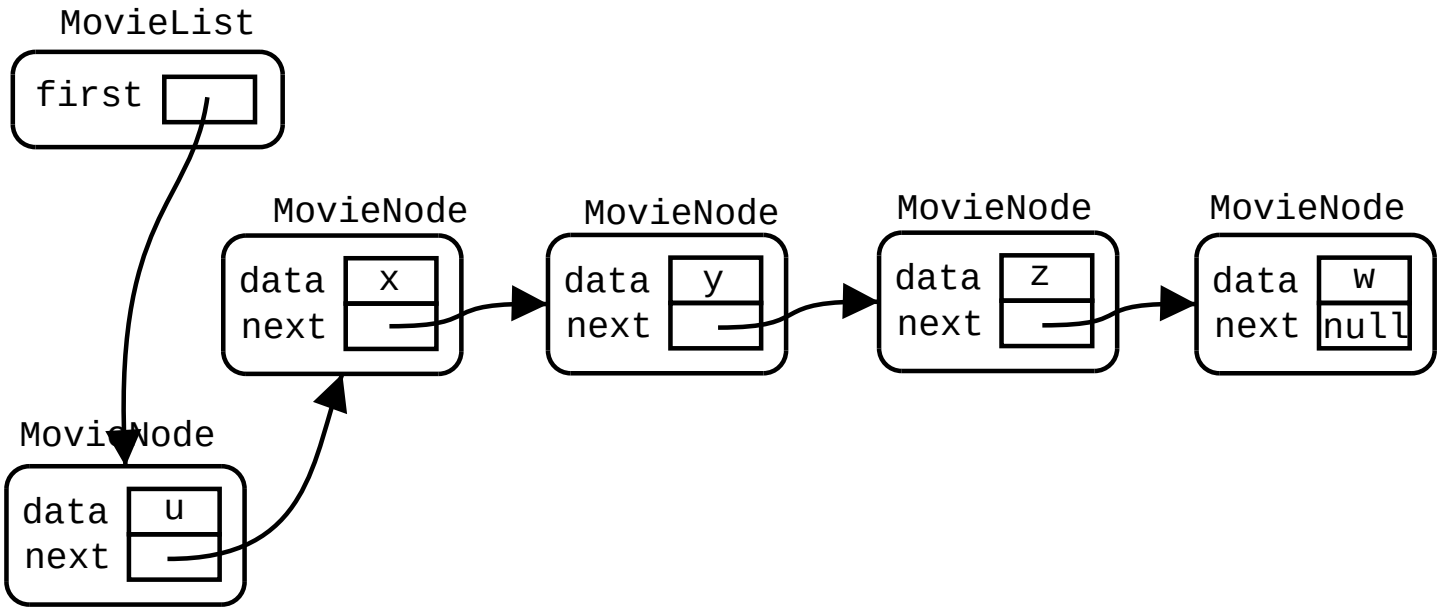
Linked Lists



Linked Lists



Linked Lists



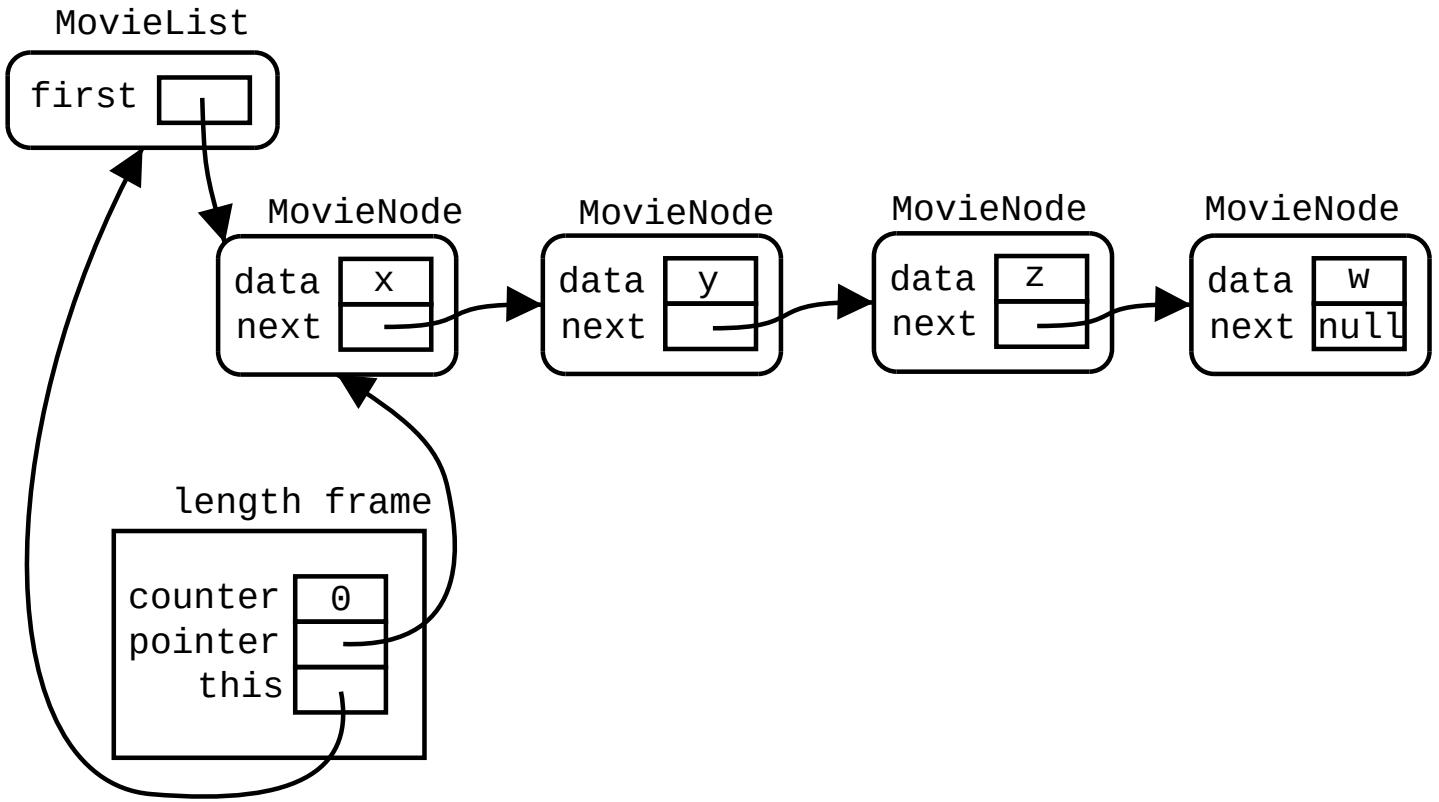
Linked Lists

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

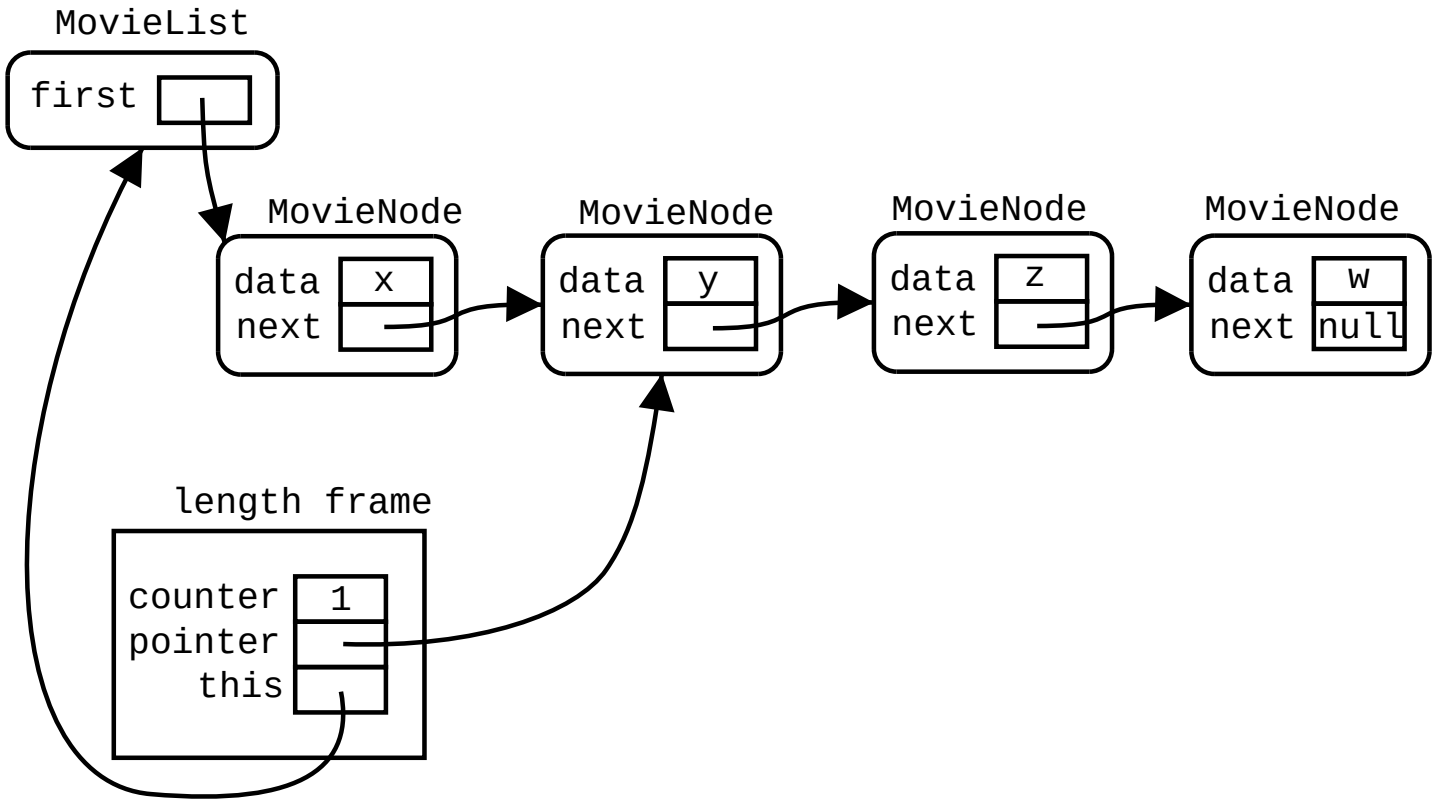
Linked Lists

```
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();
  }
}
```

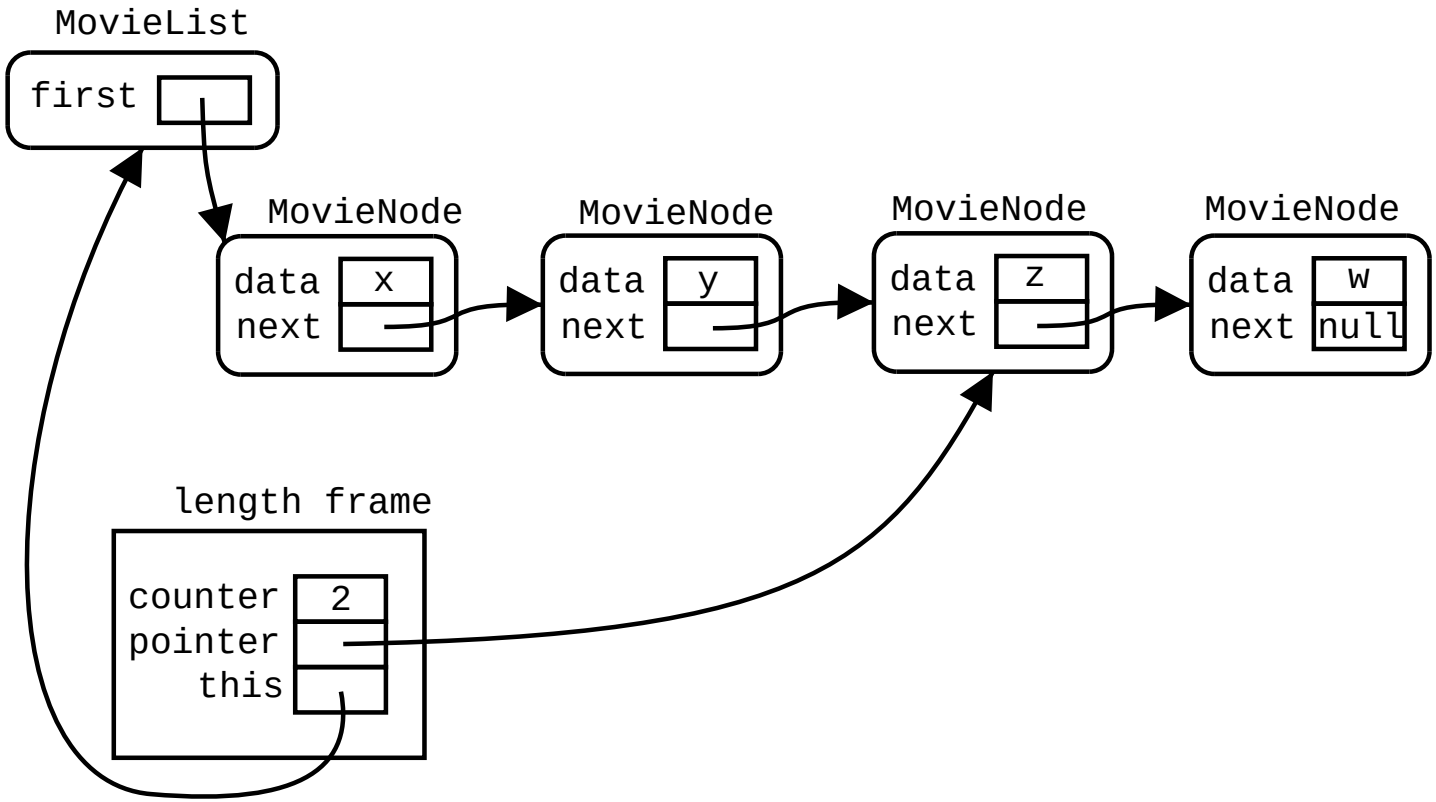
Linked Lists



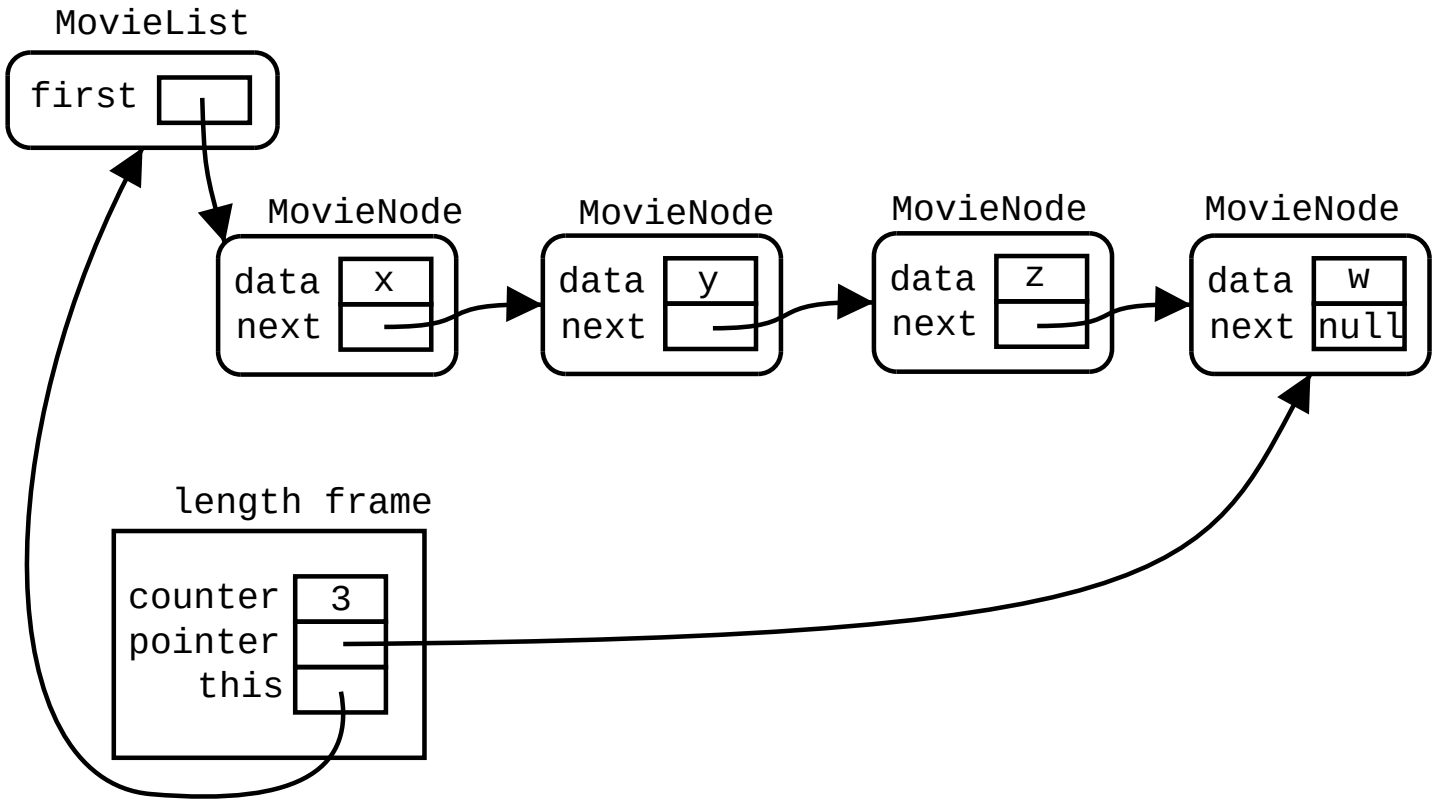
Linked Lists



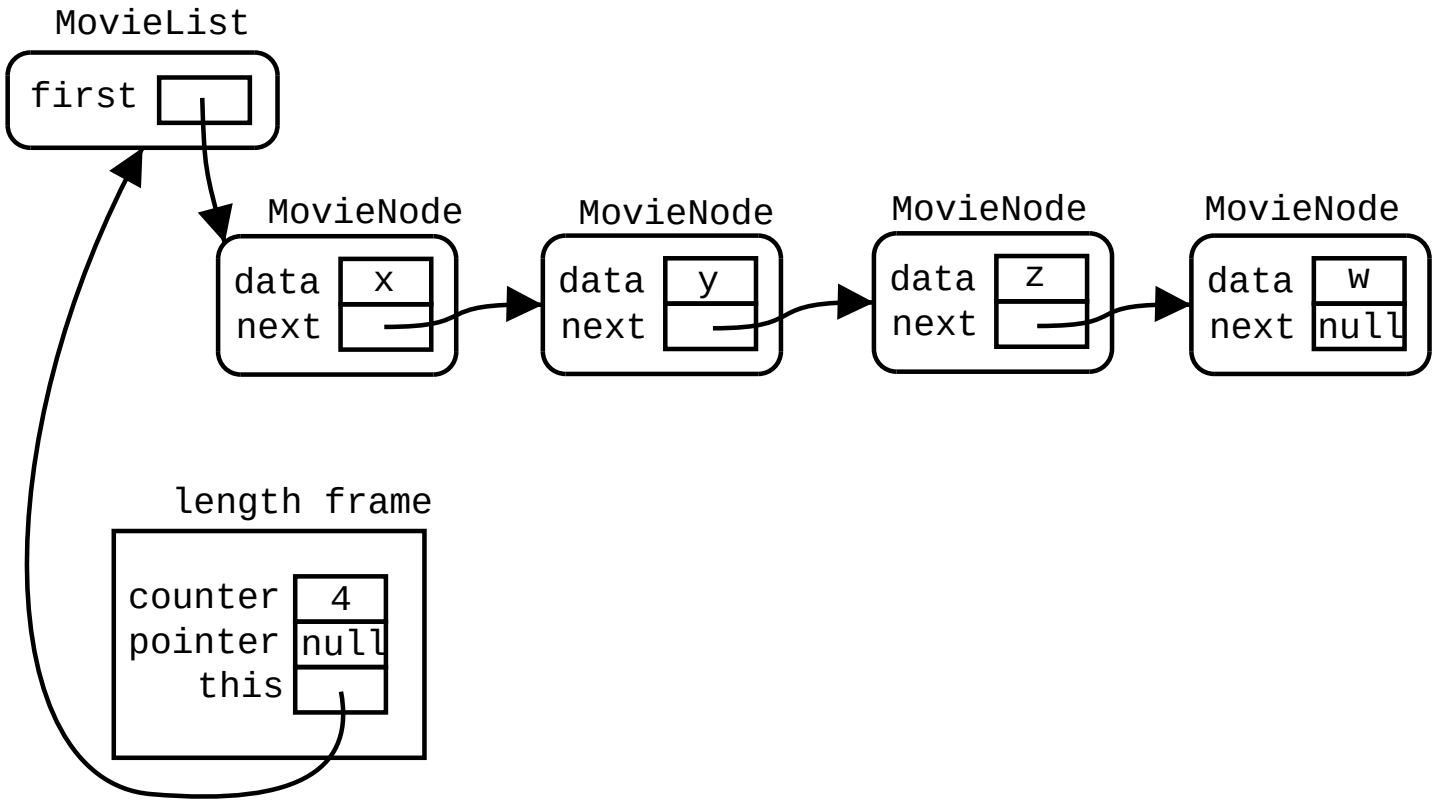
Linked Lists



Linked Lists



Linked Lists



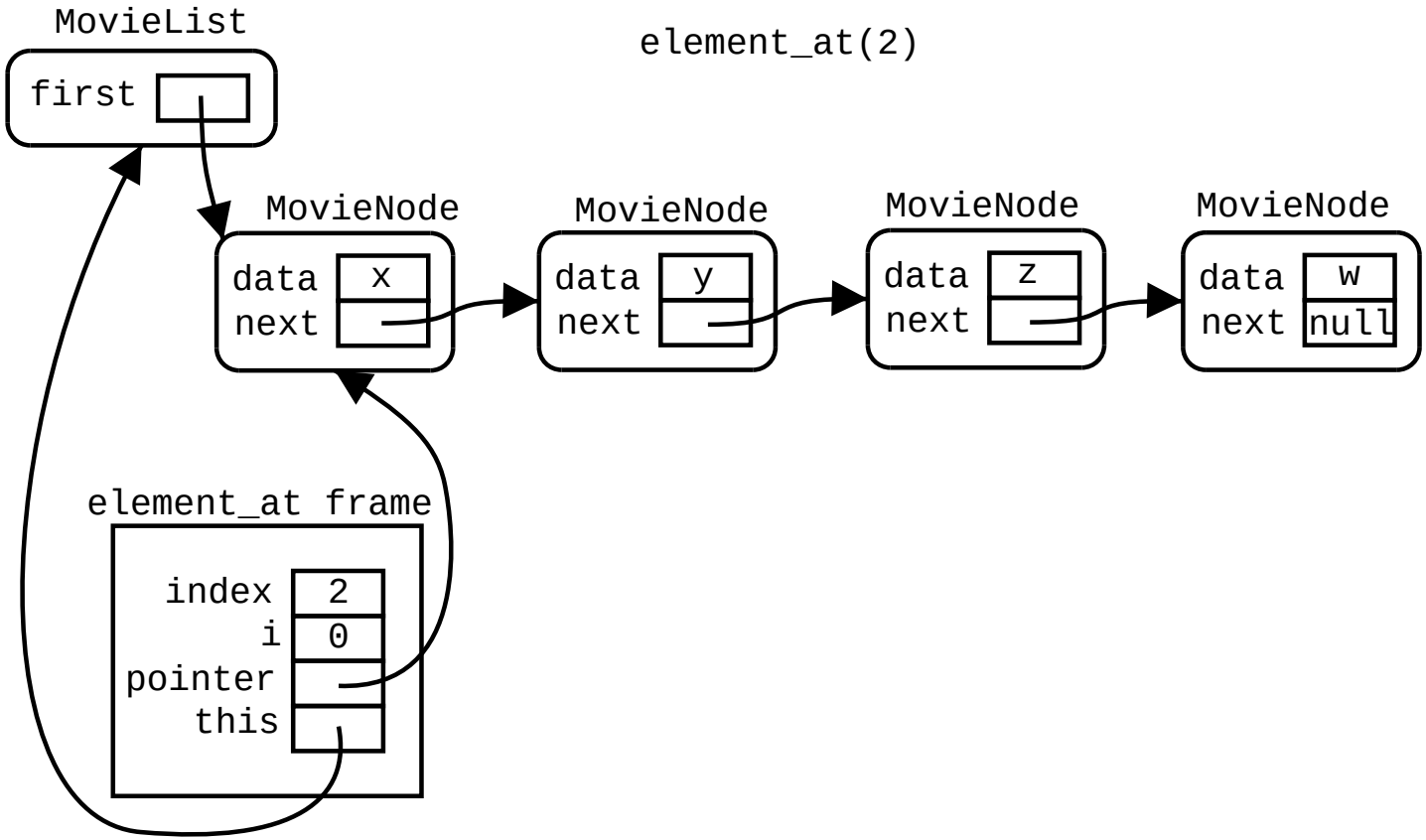
Linked Lists

```
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();
  }
}
```

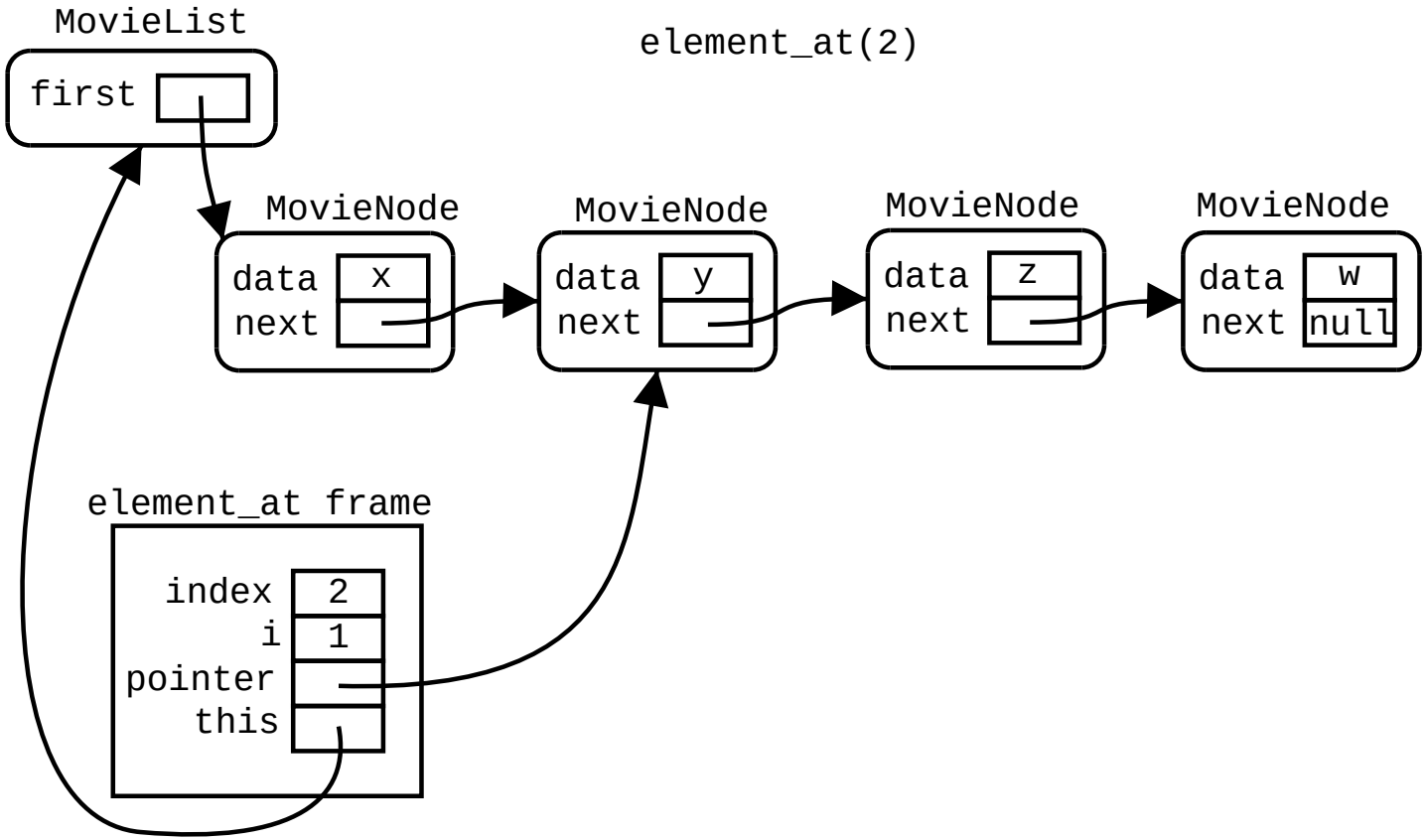
Linked Lists

```
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);
  }
}
```

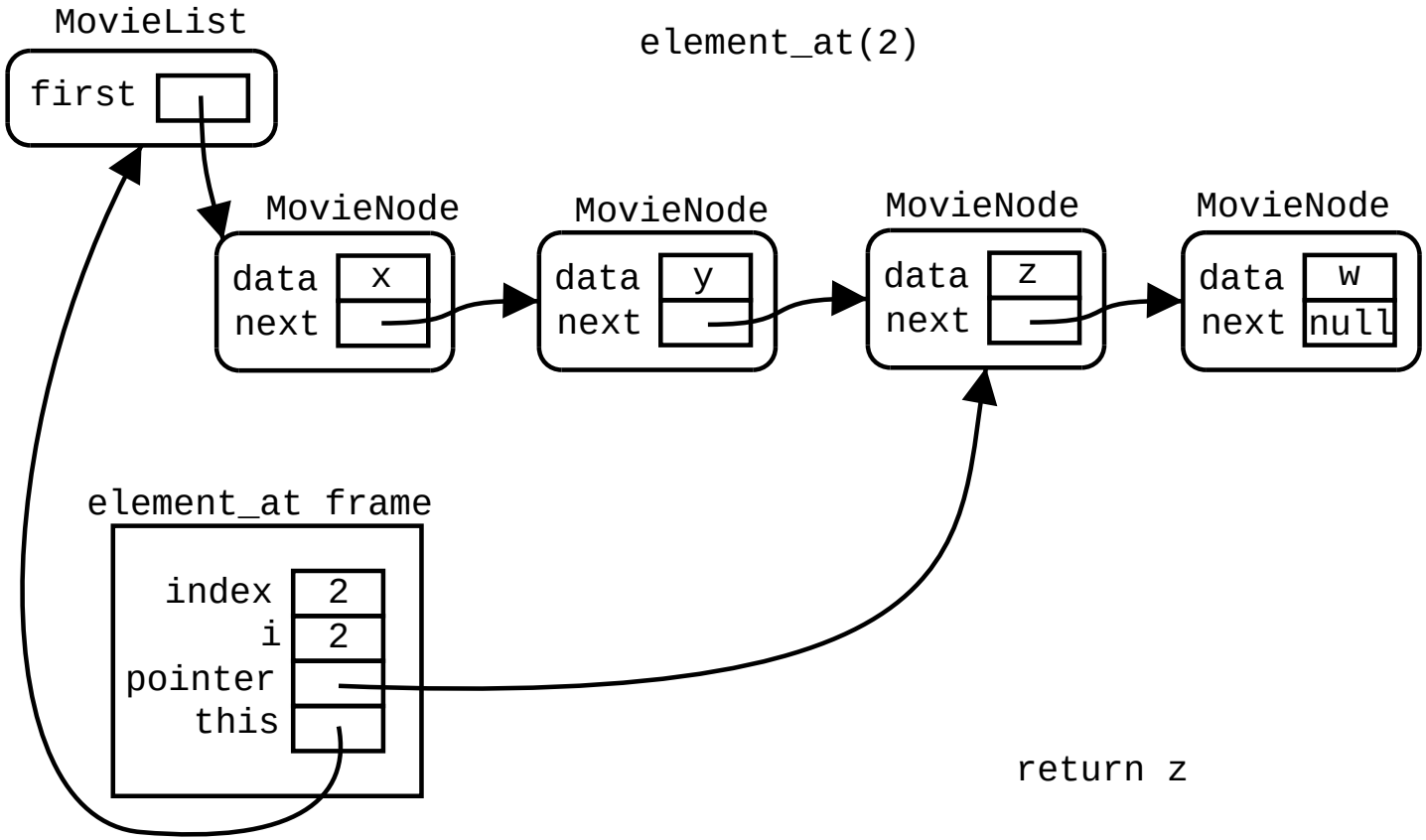
Linked Lists



Linked Lists



Linked Lists



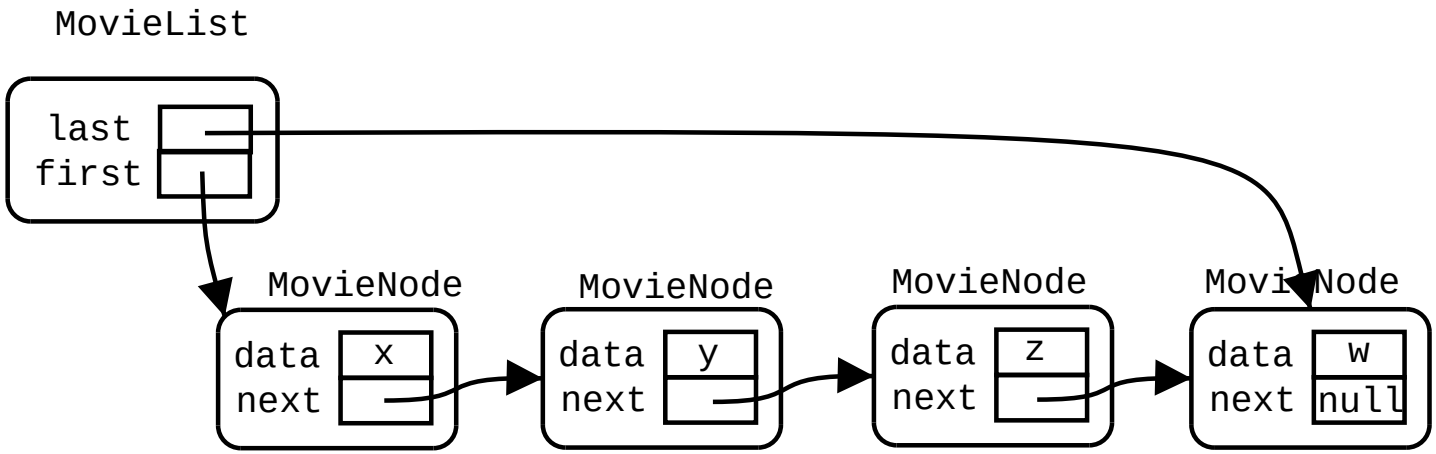
Linked Lists

```
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);
    }
  }
}
```

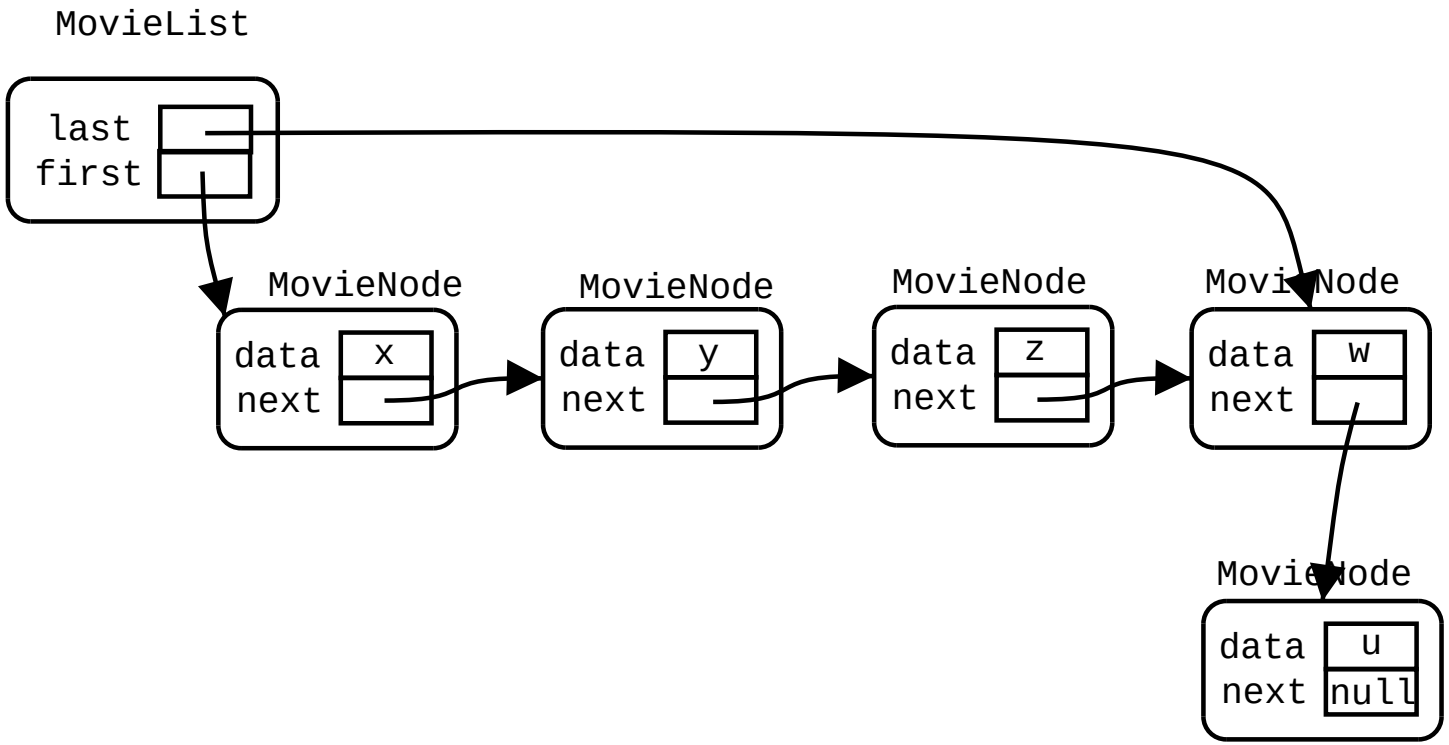
Linked Lists

```
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;
    }
  }
}
```

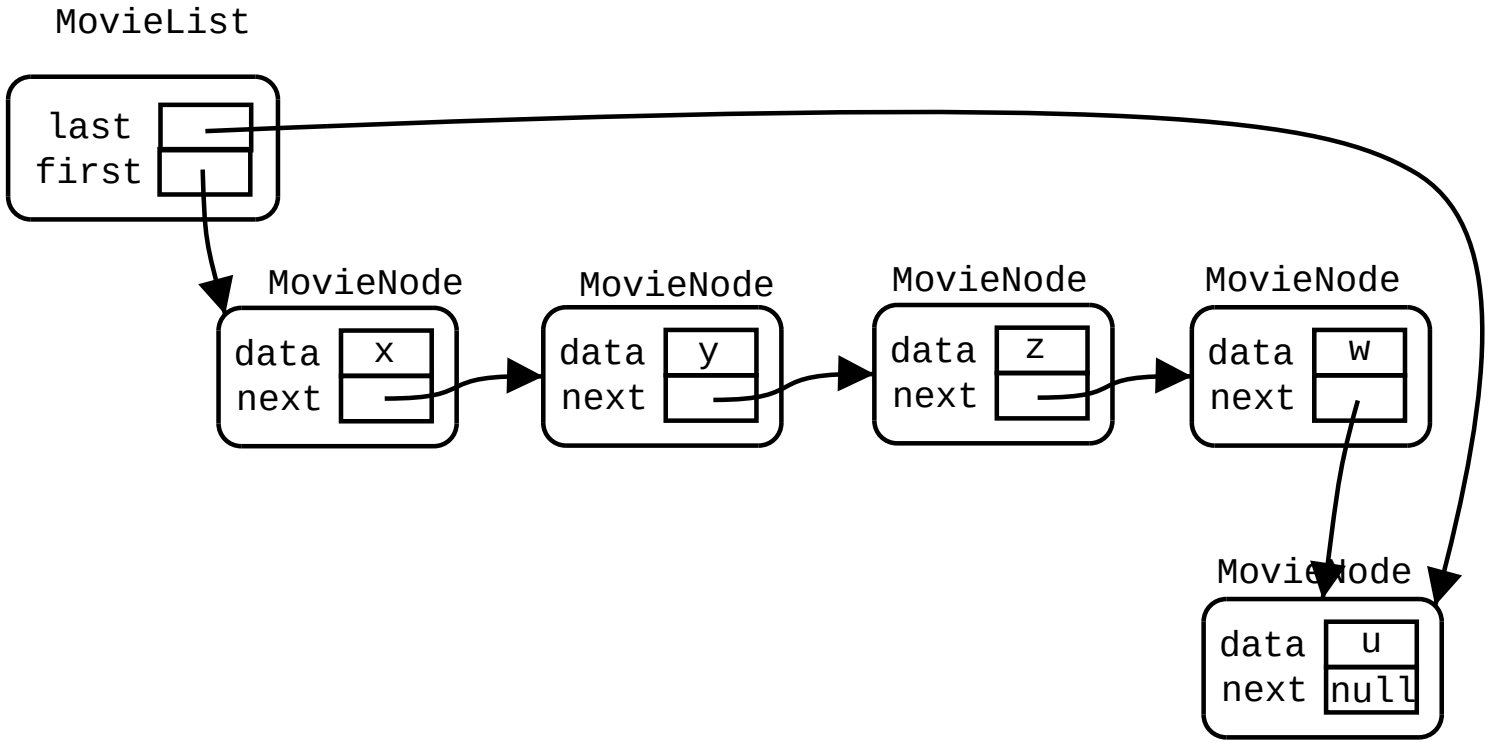
Linked-lists



Linked-lists



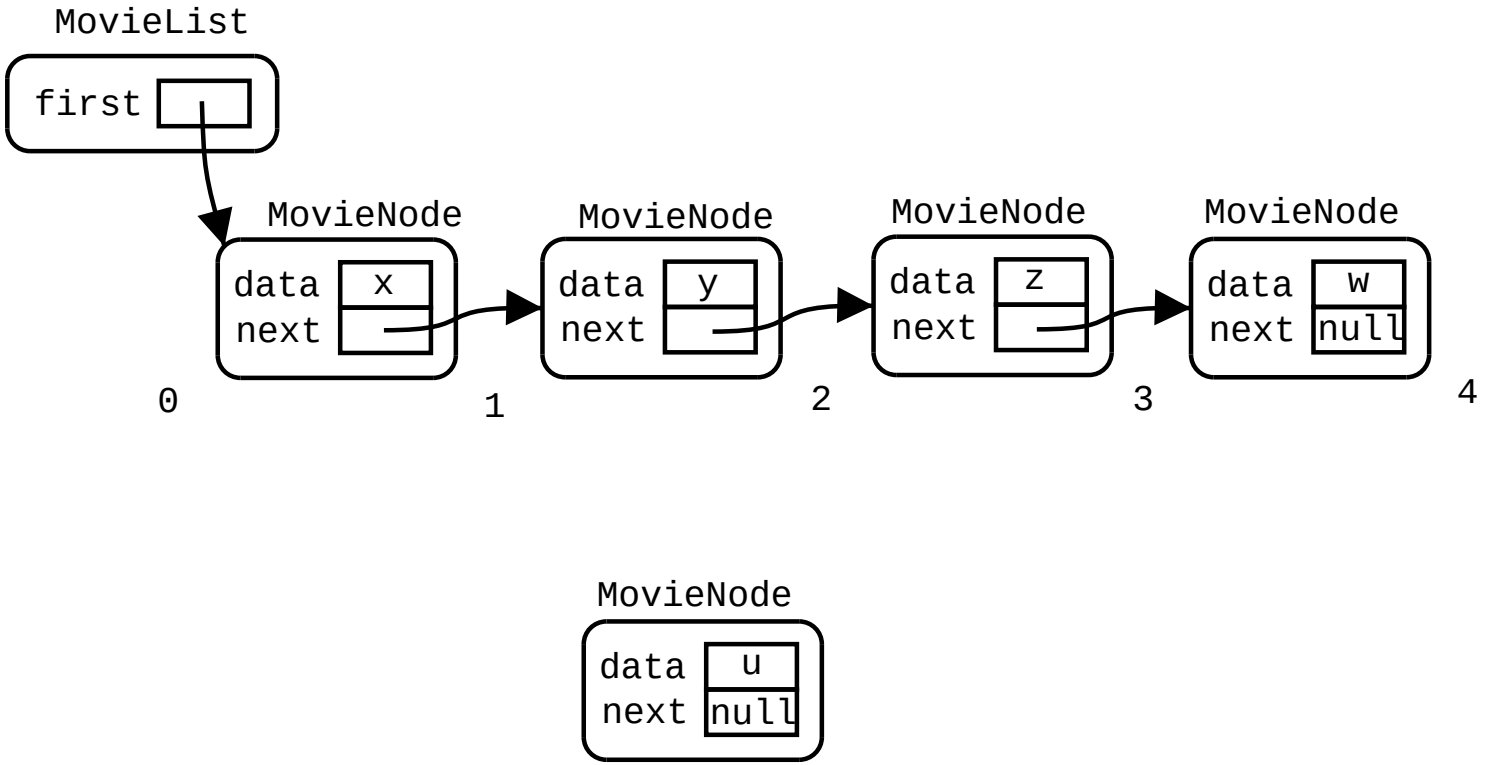
Linked-lists



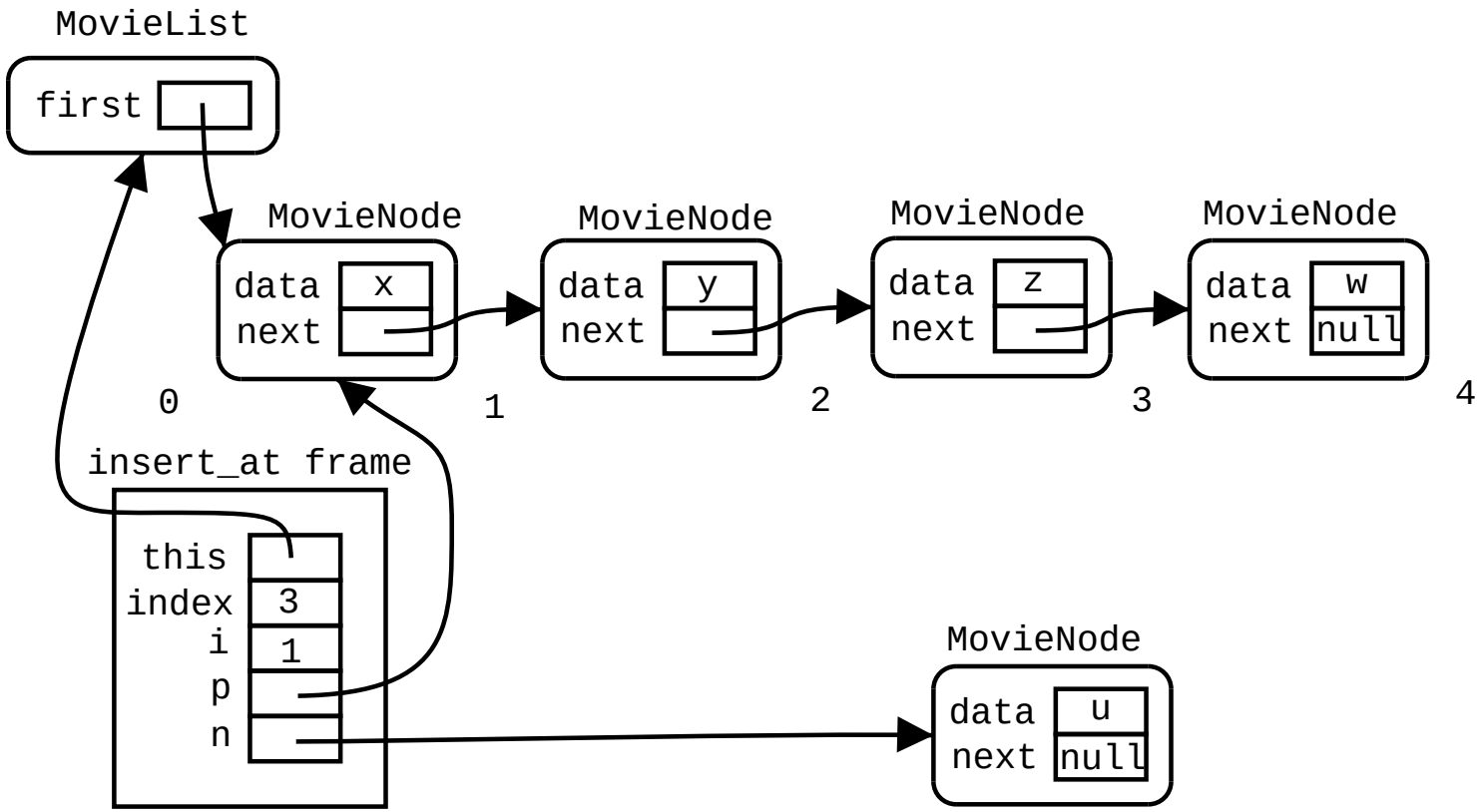
Linked-lists

```
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);
  }
}
```

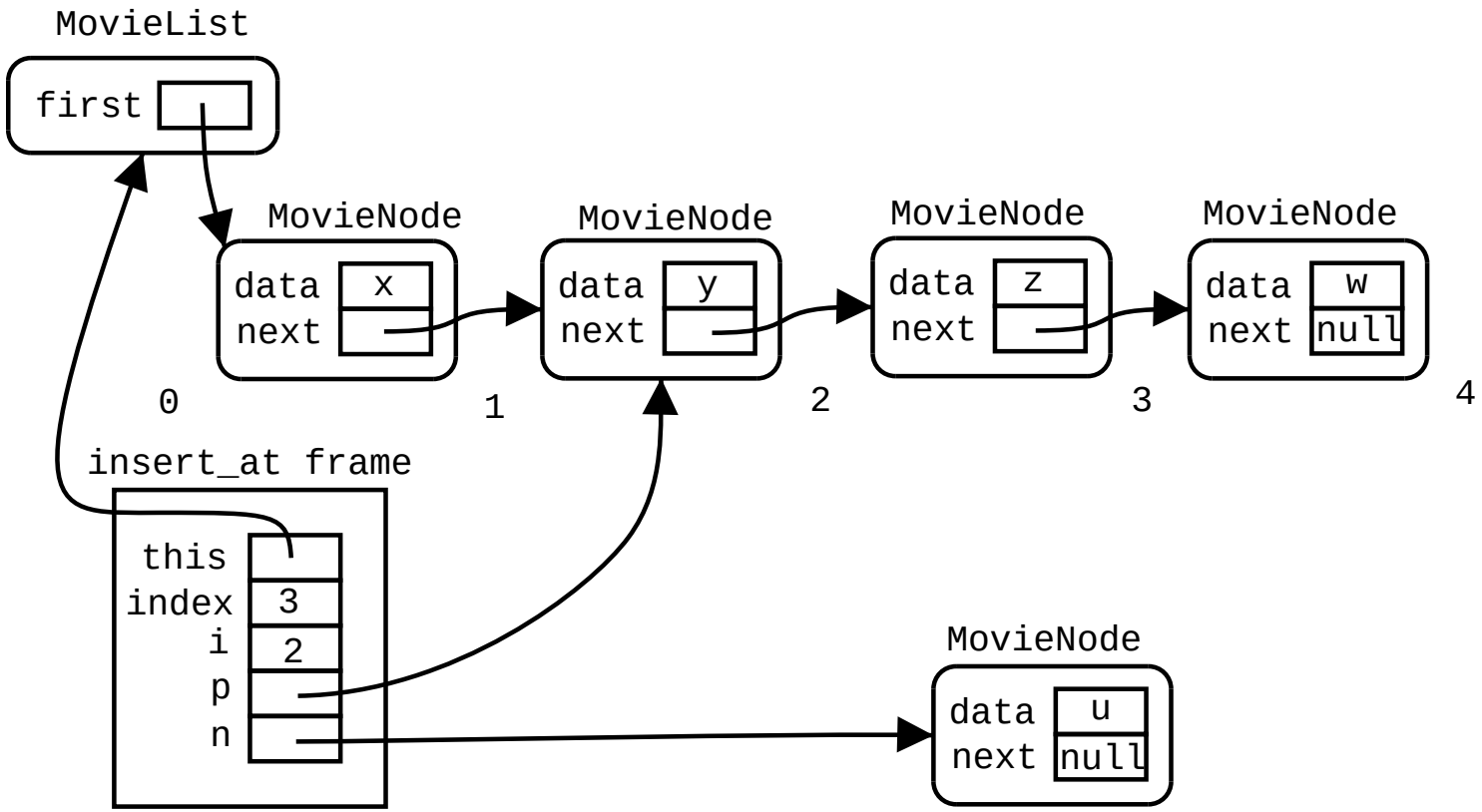
Linked-lists



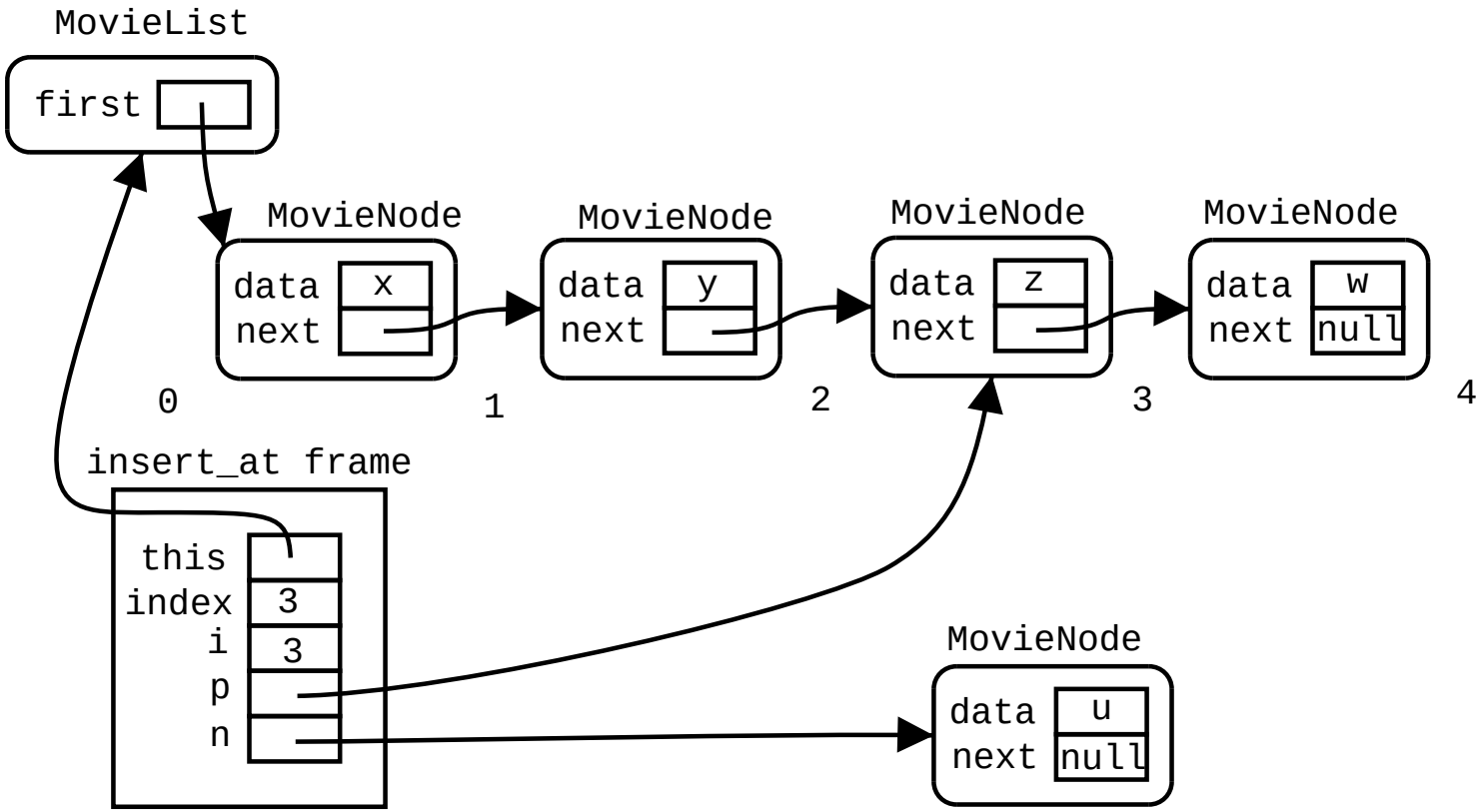
Linked-lists



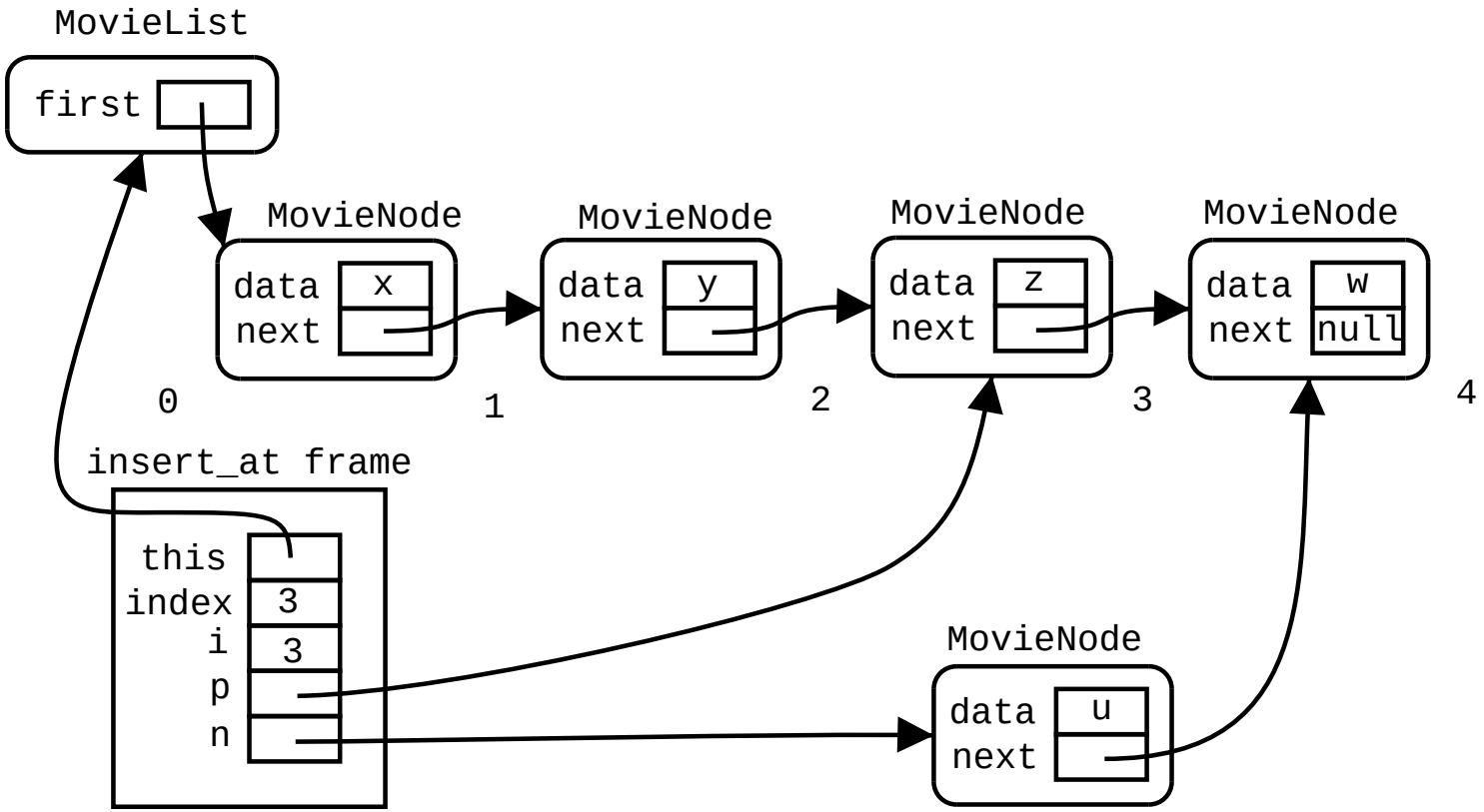
Linked-lists



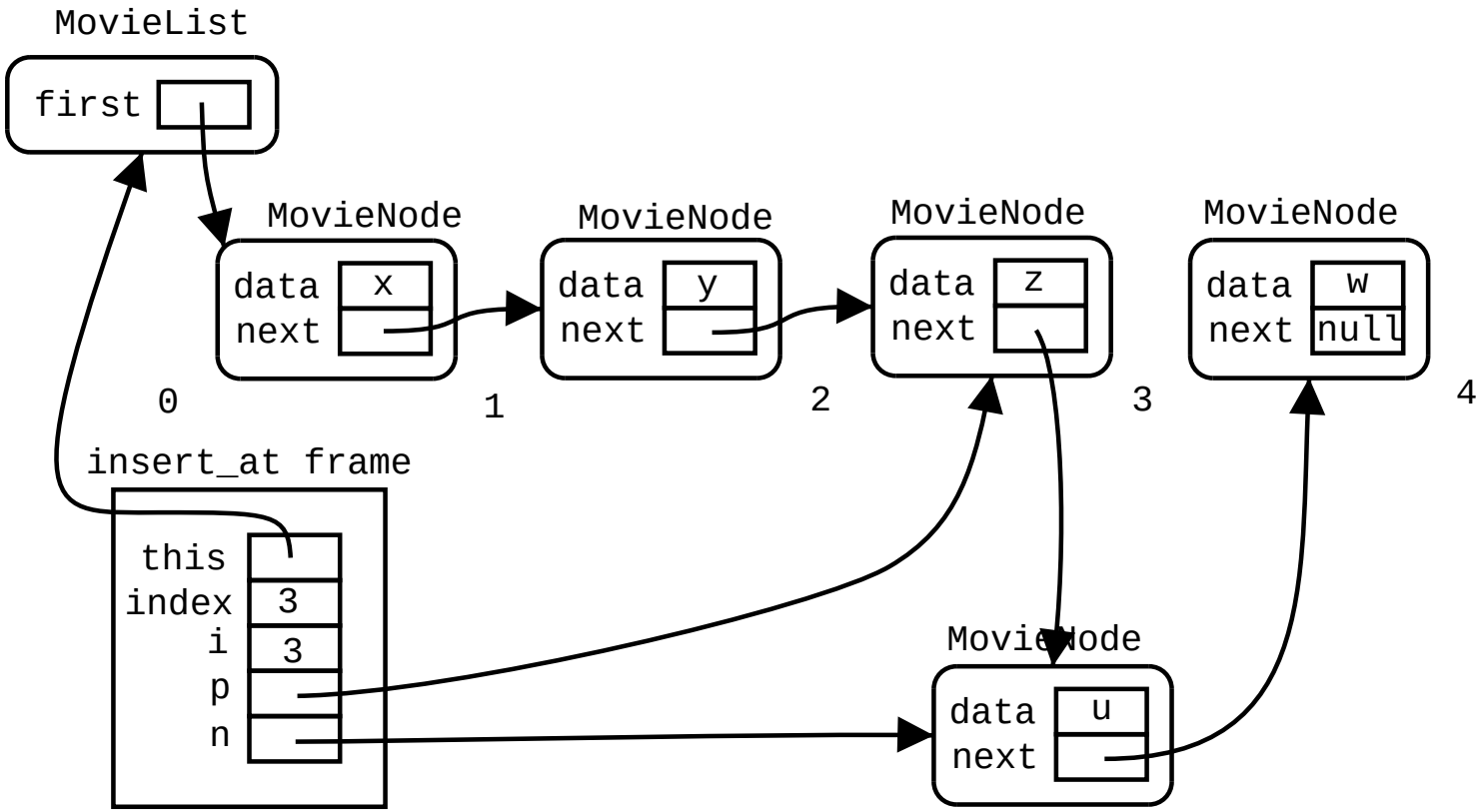
Linked-lists



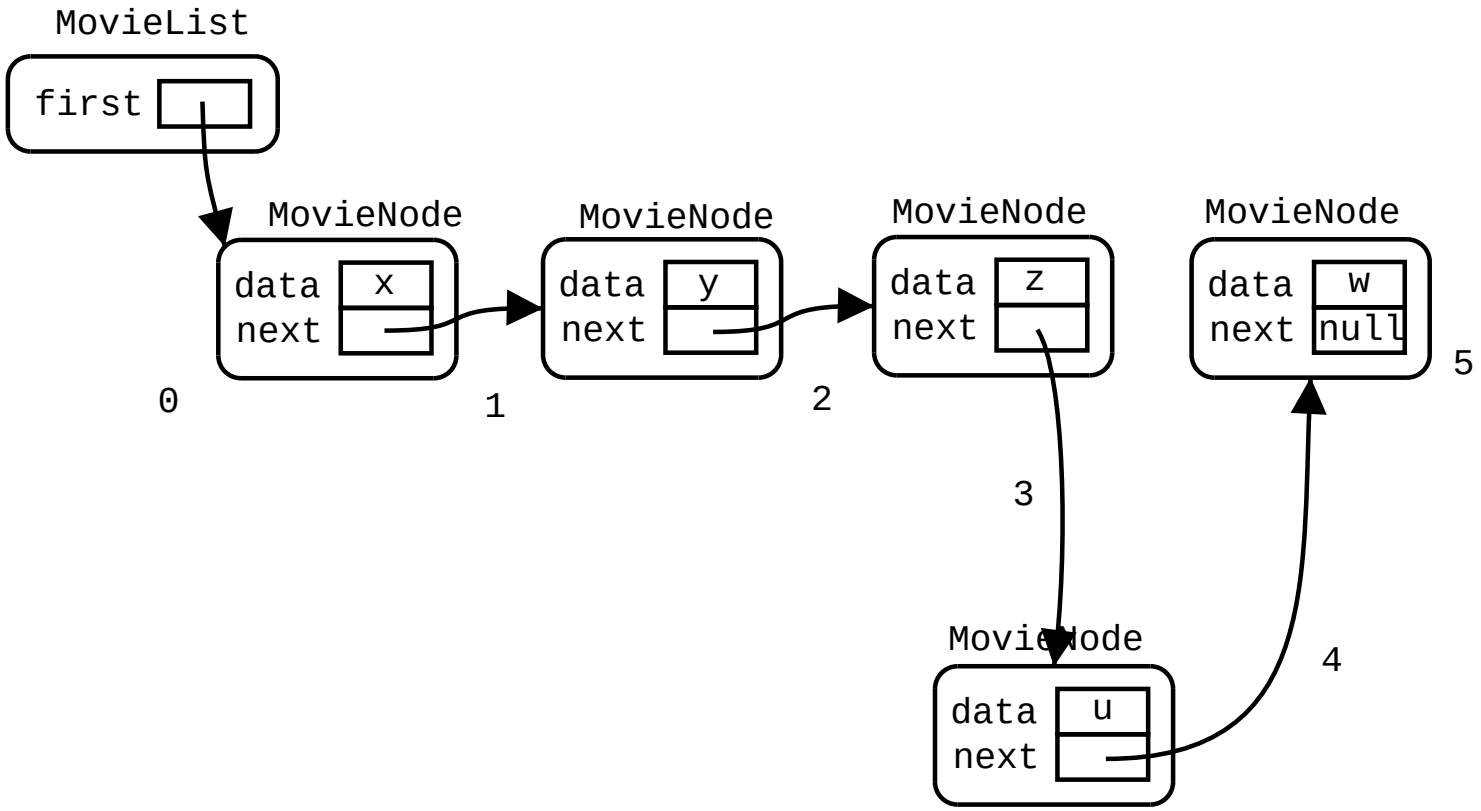
Linked-lists



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());
  }
}
```

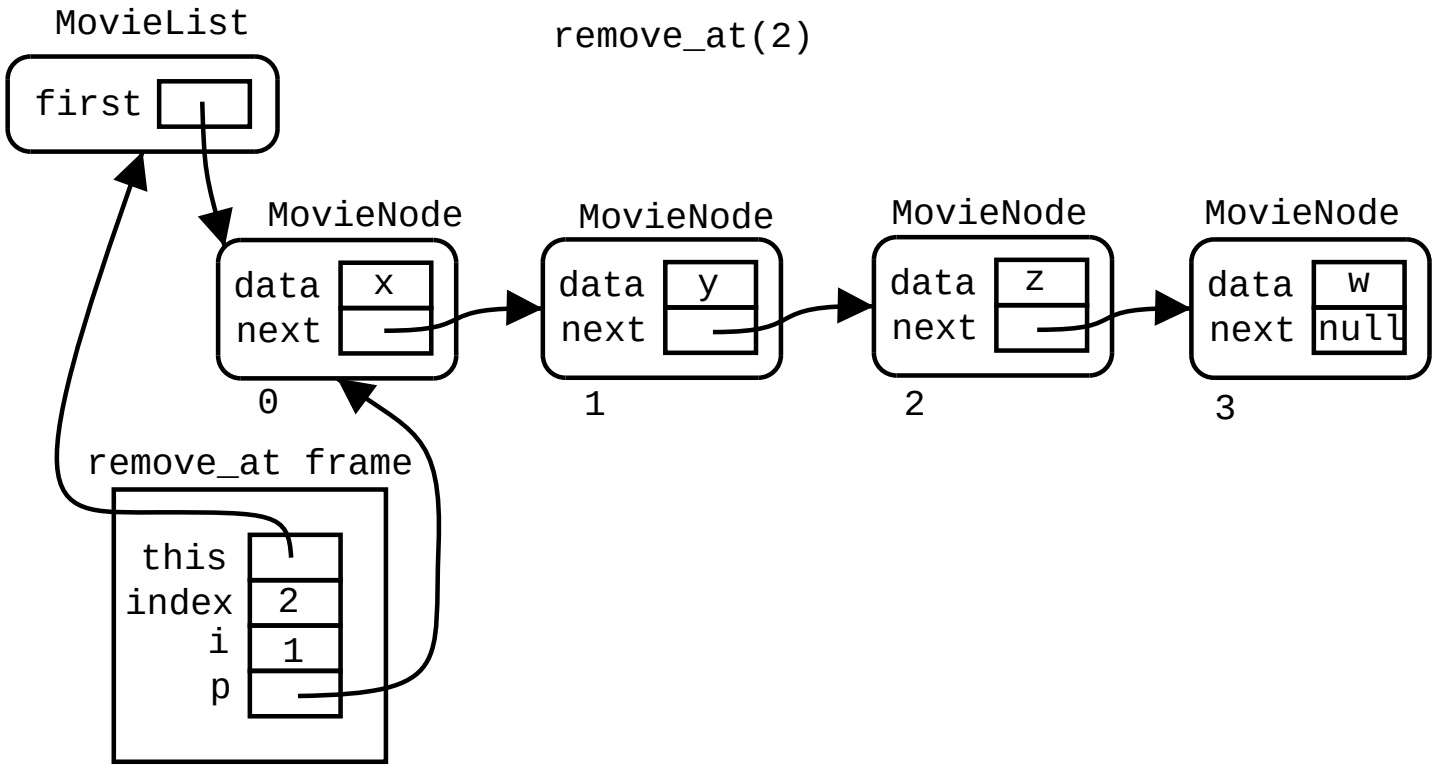
Linked-lists

```
class MovieList {
  MovieNode first;

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

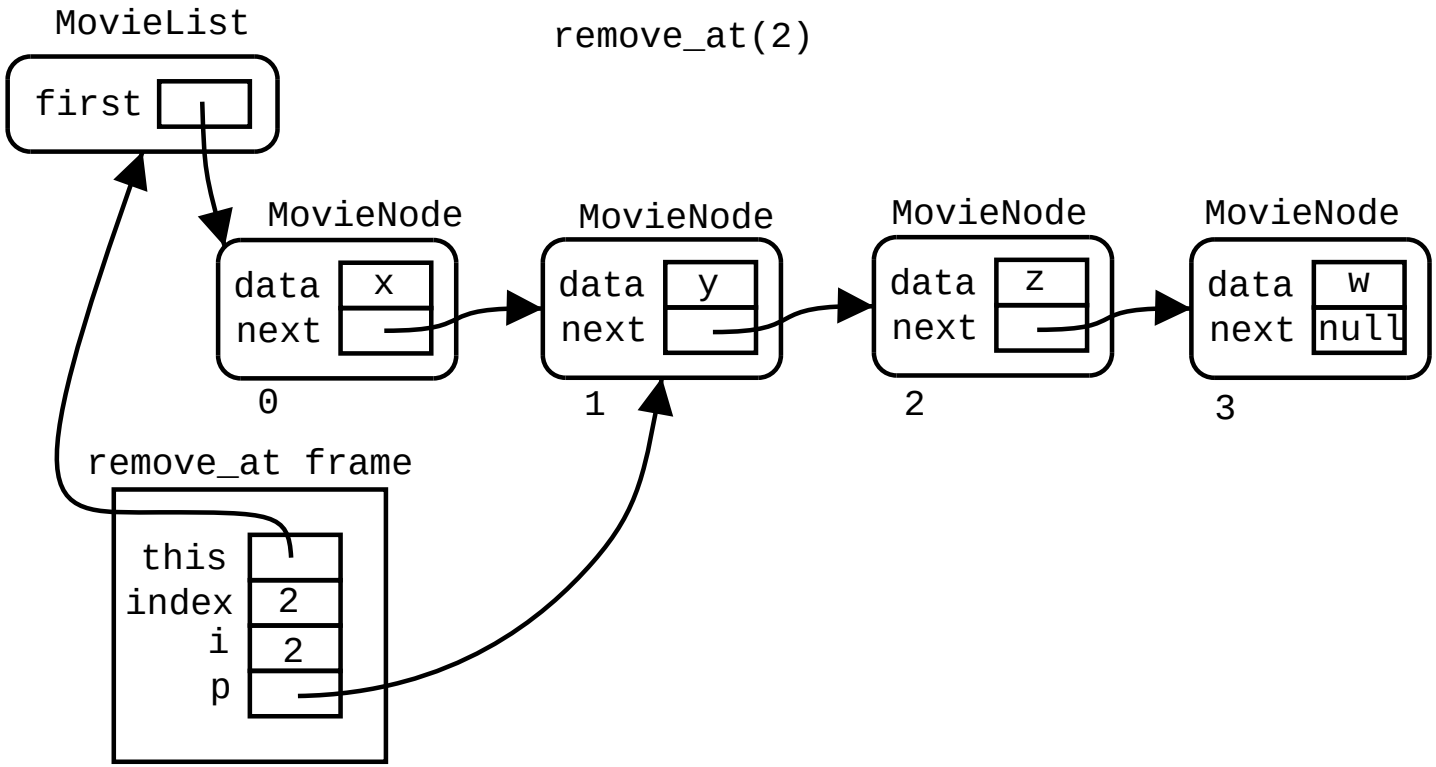
Linked-lists

```
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());
  }
}
```

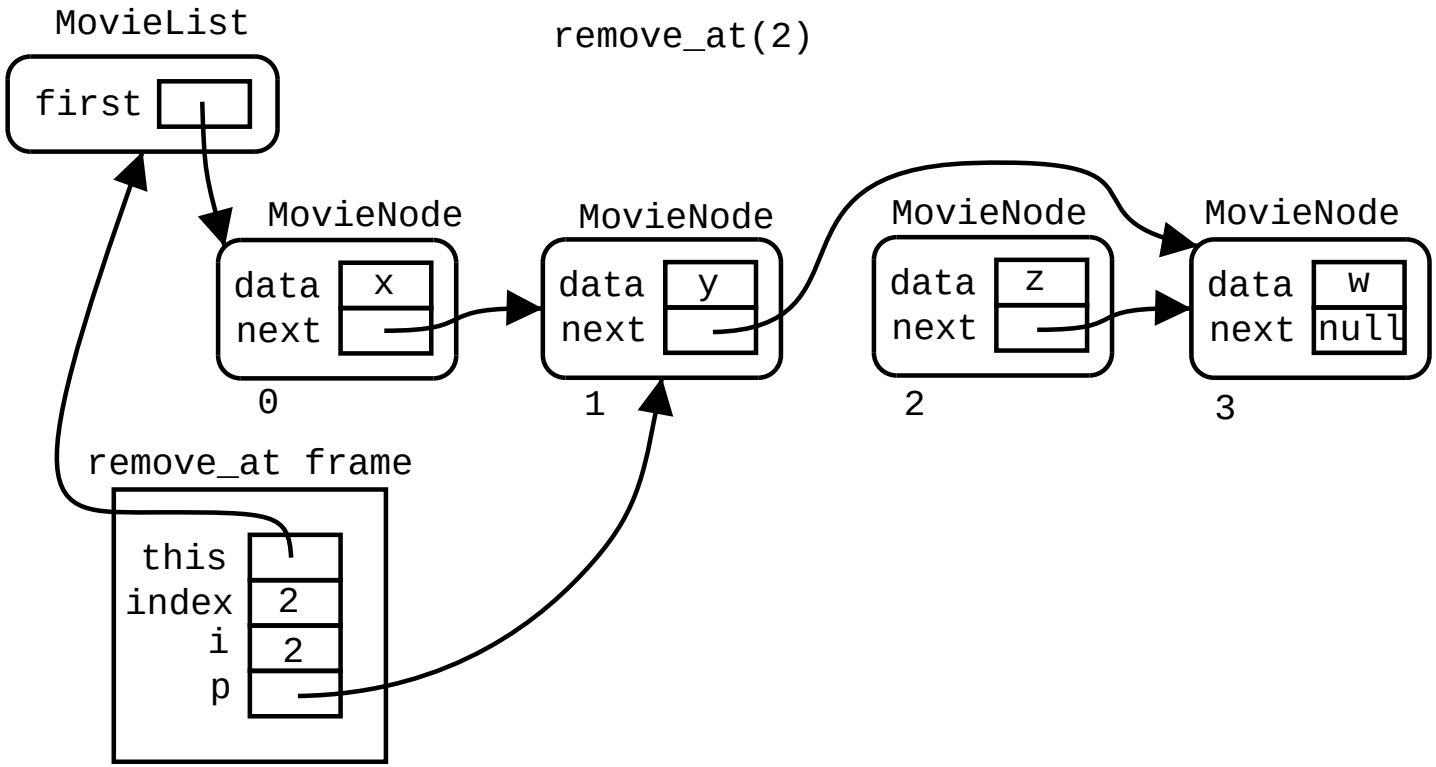
Linked-lists



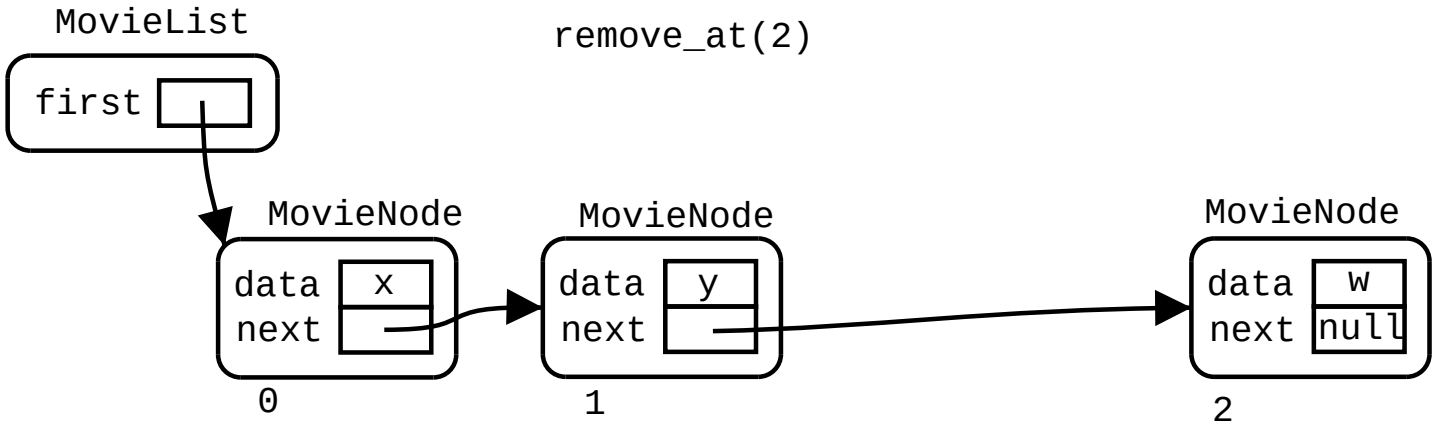
Linked-lists



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);
  }
}
```

The end