Arrays

- An array is an ordered/indexed sequence of elements of the same type.
- Array declaration

  type`[]` *variable*`;`

- Array creation:

  variable `= new` *type*`[`*integer-expression*`];`

- Array reading access:

  variable`[`*integer-expression*`]`

- Array writing access:

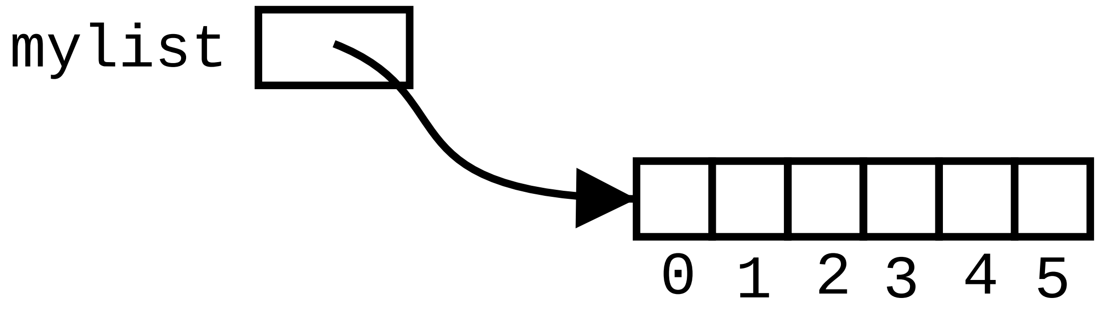
  variable`[`*integer-expression*`] =` *expression*`;`

Arrays (contd.)

- Declaring an array does not create the array itself, only a reference.
- To create an array we use the new keyword.

```
mylist = new int[6];
```

- Where the variable mylist is actually a reference to the aray itself



Example

```
double[] table;
table = new double[5];
table[0] = 3.141;
table[1] = 1.618;
table[2] = table[0] + table[1];
table[4] = table[2];
table[3] = 1;
table[0] = 1.414;
int i = 0;
while (i < table.length) {
    System.out.println(table[i]);
    i++;
}
```

Processing arrays

- Processing arrays is a generalization of processing strings.
- `a[i]` is analogous to `s.charAt(i)`, but only for reading the `i`-th, not for writing: `charAt` cannot be used for modifying a string. This is: `s.charAt(i) = expr`; is illegal syntax.
- Use loops to traverse an array.
- The length of an array `a` can be obtained by the expression `a.length`
- This is independent of the number of slots that hold a value

Example 1

- Filling an array

```
static void fill(double[] a)
{
  int index;
  index = 0;
  while (index < a.length) {
    a[index] = Math.random();
    index++;
  }
}
```

Example 2

- Finding the minimum number in an array

```
static double find_min(double[] a)
{
  int index;
  double minimum;
  index = 0;
  minimum = 999999999.9;
  while (index < a.length) {
    if (a[index] < minimum) {
      minimum = a[index];
    }
    index++;
  }
  return minimum;
}
```

Example 3

- Returning the index where the minimum is located

```
static int find_min(double[] a)
{
  int index, min_index;
  double minimum;
  index = 0;
  min_index = 0;
  minimum = a[0];
  while (index < a.length) {
    if (a[index] < minimum) {
      minimum = a[index];
      min_index = index;
    }
    index++;
  }
  return min_index;
}
```

Initializing arrays

- If we have a class

```
class B {
  int n;
  B(int x) { n = x; }
}
```

- and somewhere else we declare and create an array

```
B[] list = new B[7];
```

- Then all the slots in the array will be initialized to `null`. This is, the constructor for `B` will not be called. If we want an object created in each slot, we have to do it explicitely:

```
for (int i=0; i < list.length; i++)
  list[i] = new B(3);
```

Initializing arrays

- Arrays can be initialized with default values using the syntax:

  type[] *var* = { *expr1*, *expr2*, ..., *exprn* };

  Where each *expri* is of type *type*.

- For example:

```
int[] a = { 1, 1, 2, 3, 5 };
Z[] u = { new Z(), new Z() };
```

Processing arrays: safety

- Since arrays are references, it is often useful to check whether they are null or not before using them, to avoid null-pointer exceptions.
- If the array has as base type a class, it is also useful to check that each slot which will be processed or accessed is not null.
- For example:

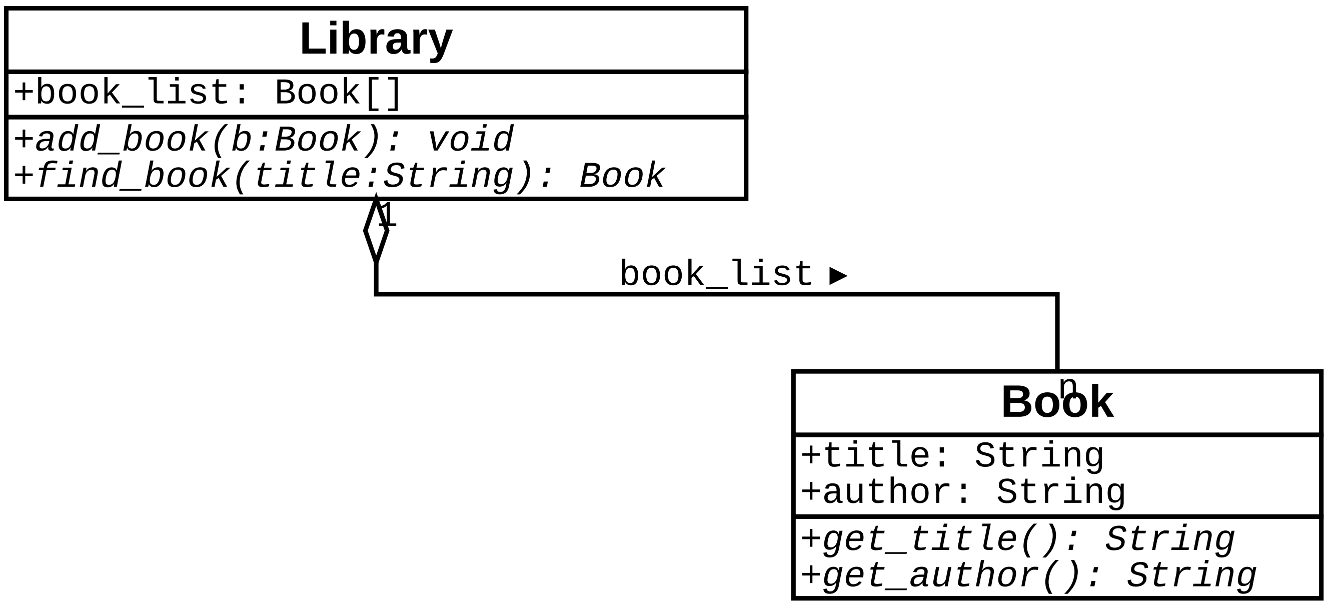
```
class A { int x; }
class B {
  static void m(A[] list)
  {
    if (list != null) {
      for (int i = 0; i < list.length; i++) {
        if (list[i] != null) {
          list[i] = 2 * i;
        }
      }
    }
  }
```

Array applications

- Library: Book database
- Problem: Create a database of books, which supports the operations of adding a new book, and searching for a book by title.
- Analysis:
  - Identify objects and classes:
    - Individual books
    - A library: book database
  - Relationships
    - Each book *has a* title and an author
    - A book database *has a* list of books
  - Operations/Interactions/Behaviour
    - Adding books to a database
    - Searching for a book in a database

Array applications (contd.)

- Design
  - Class diagram



Array applications (contd.)

```
class Book {
  private String title, author;
  public Book(String t, String d)
  {
    title = t;
    author = d;
  }
  public String title() { return title; }
  public String author() { return author; }

}
```

Array applications (contd.)

```
class Library {
  private Book[] book_list;
  private int next_available;
  public int number_of_books;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    next_available = 0;
    number_of_books = 0;
  }

  // Continues below...
```

```
public void add_book(Book m)
{
  if (next_available < book_list.length) {
    book_list[next_available] = m;
    next_available++;
    number_of_books++;
  }
}
```

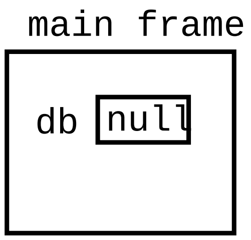
```
  public Book find_book(String title)
  {
    Book b;
    String t;
    int index = 0;
    while (index < number_of_books) {
      b = book_list[index];
      t = b.title();
      if (t.equals(title)) {
        return b;
      }
      index++;
    }
    return null;
  }
} // End of Library
```

Array applications (contd.)

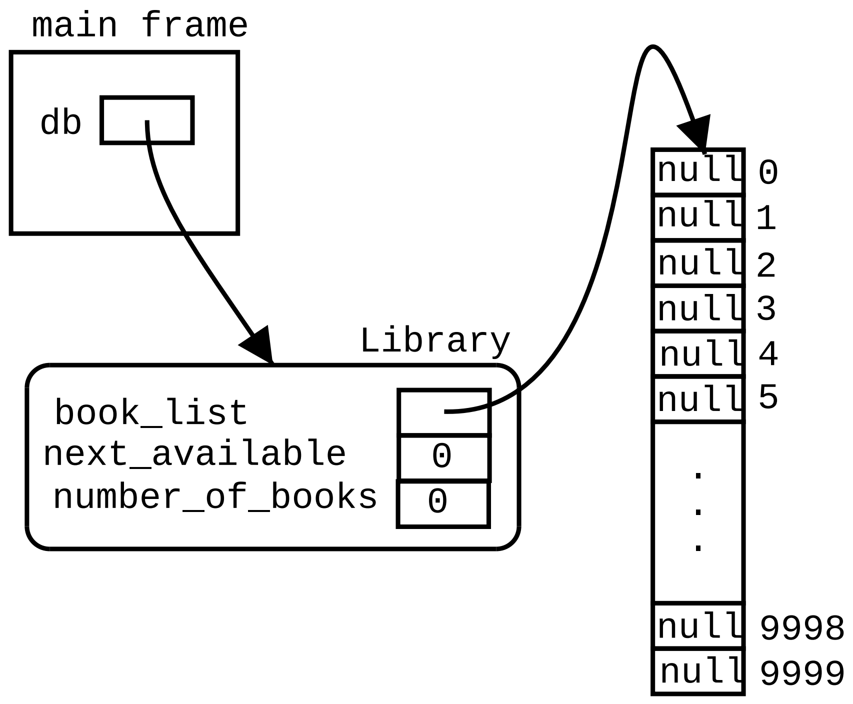
- Using the database

```
public class Test {
  public static void main(String[] args)
  {
    Library db = new Library(10000);
    Book m;
    m = new Book(“Fictions”,“Borges”);
    db.add_book(m);
    m = new Book(“Hamlet”,“Shakespeare”);
    db.add_book(m);
    m = new Book(“L\’Avare”,“Moliere”);
    db.add_book(m);
    Book k = db.find_book(“Hamlet”);
    System.out.println(k.author());
  }
}
```

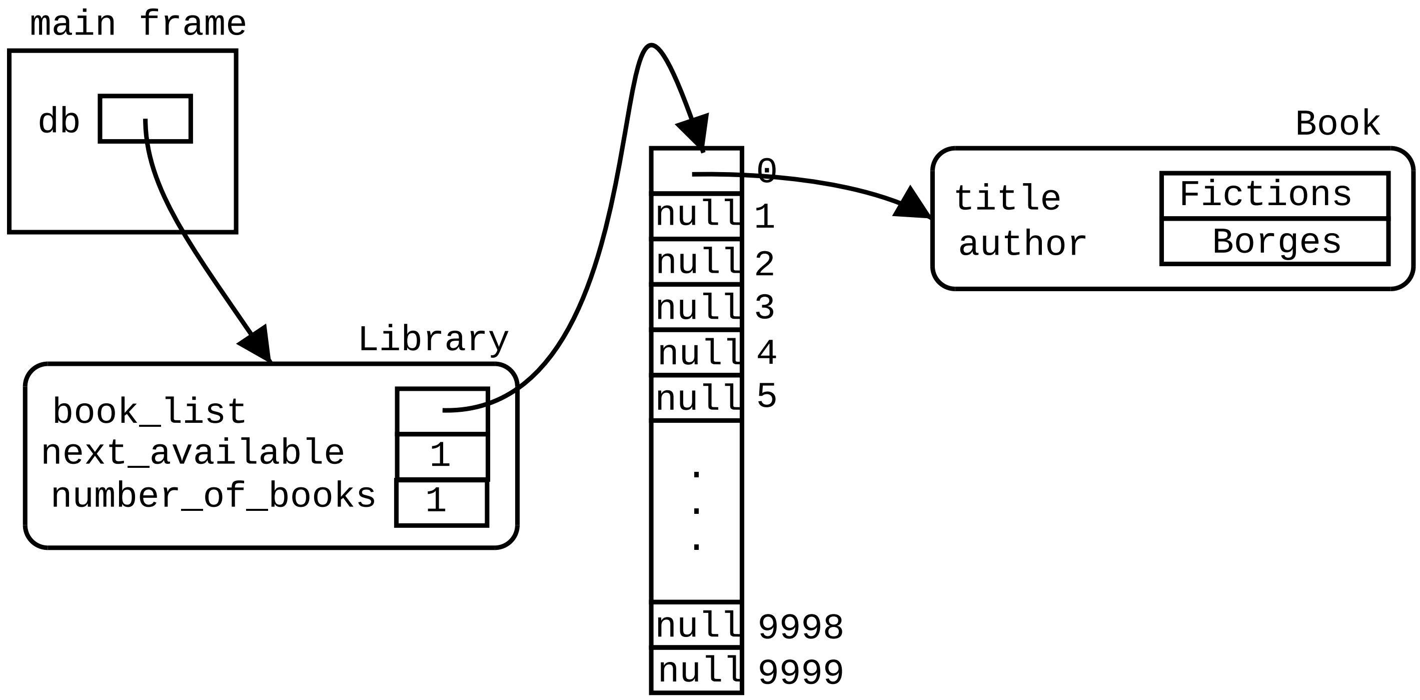
Array applications (contd.)



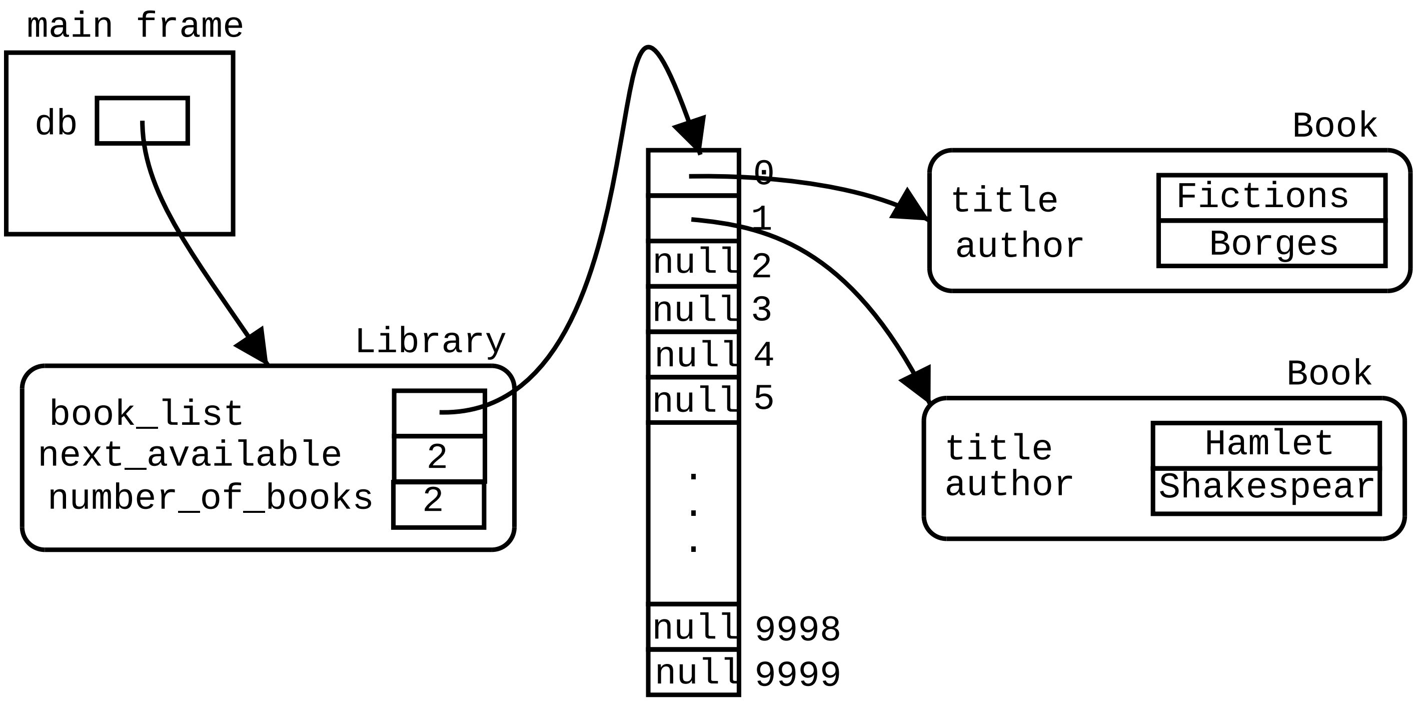
Array applications (contd.)



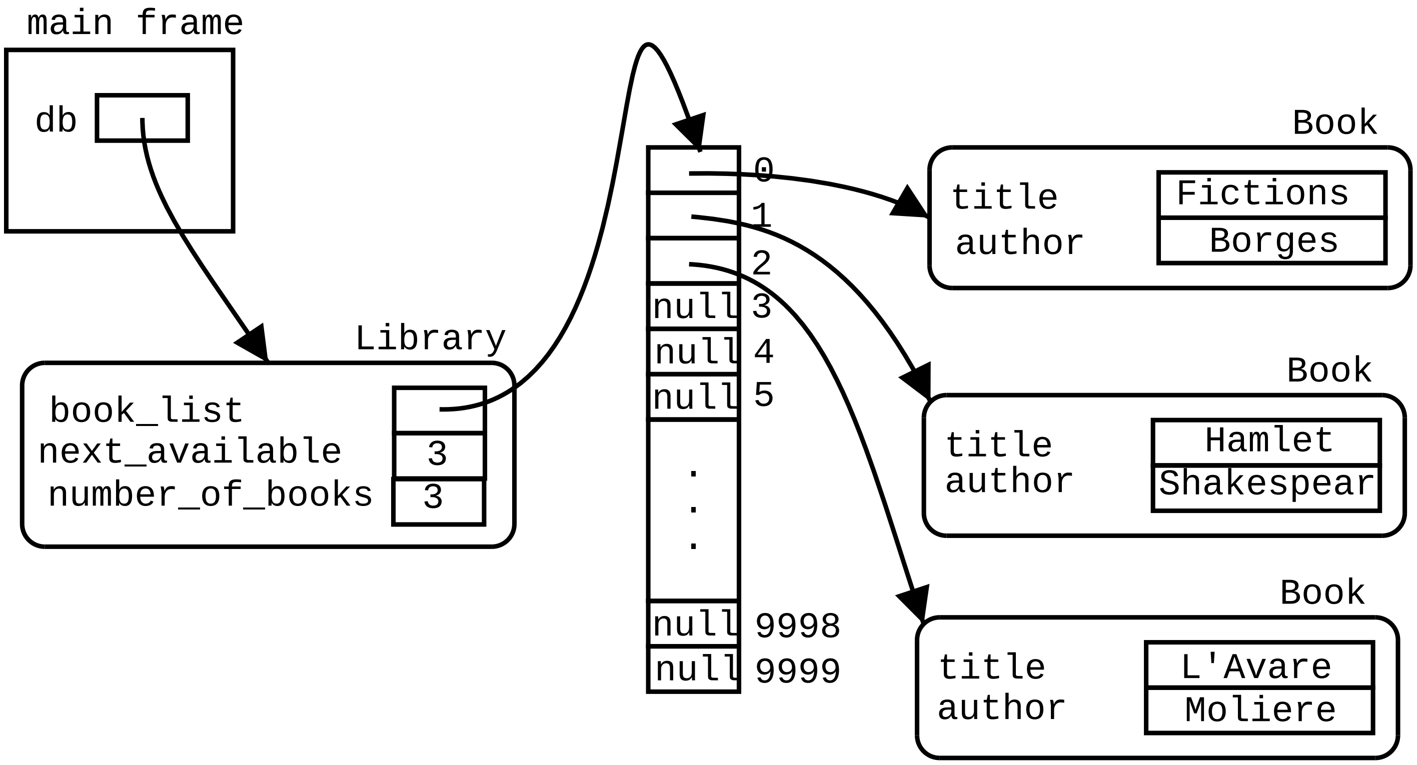
Array applications (contd.)



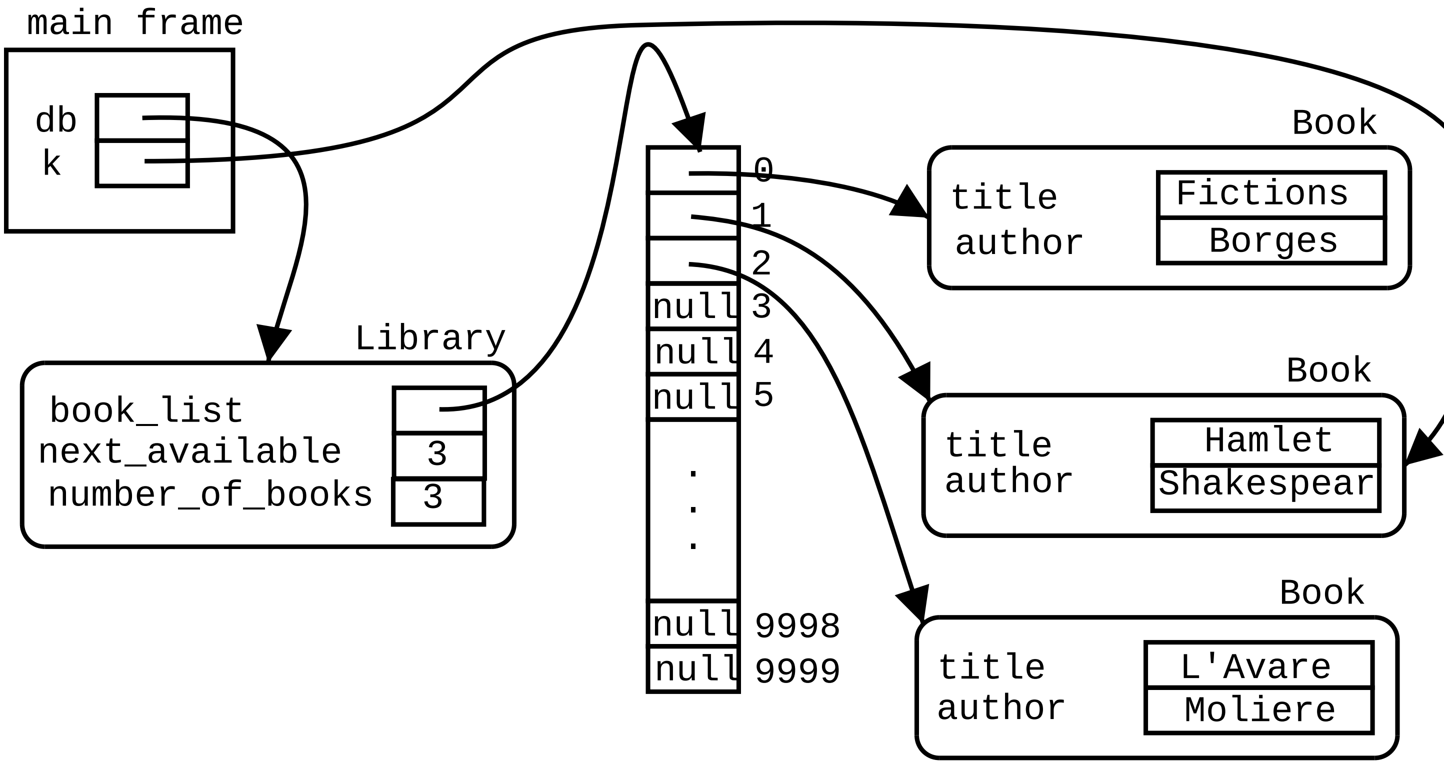
Array applications (contd.)



Array applications (contd.)



Array applications (contd.)



Array applications (contd.)

- Deleting elements from the database:
- To delete a book with title t:

1. Find the index i where the book with title t is located
2. Set book_list[i] to null

Array applications (contd.)

- Deleting elements from the database:
- To delete a book with title t:

1. Find the index i where the book with title t is located
2. If i is a legal index:
   (a) Set book_list[i] to null

Array applications (contd.)

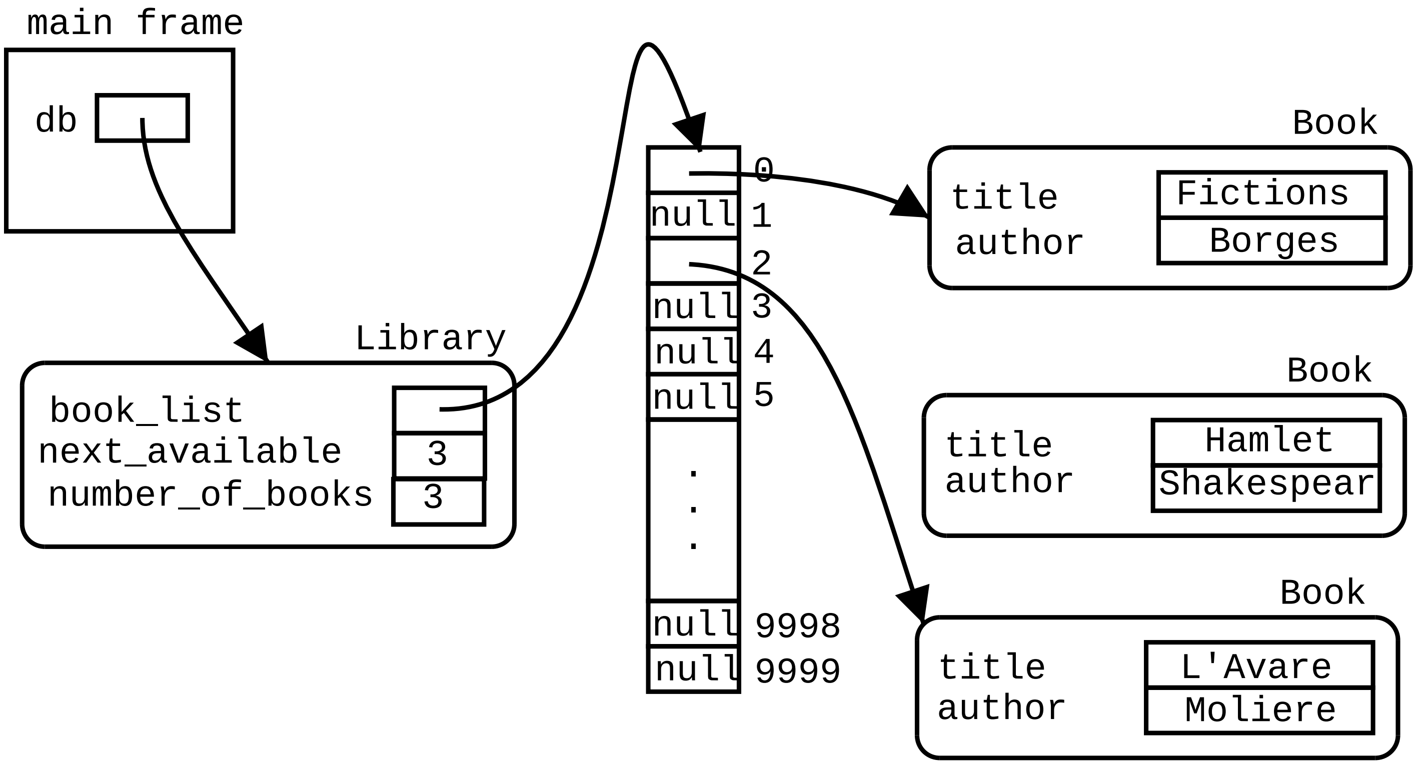
```
public int book_index(String title)
{
  for (int i=0; i < book_list.length; i++) {
    Book m = book_list[i];
    if (m != null) {
      String s = m.title();
      if (s.equals(title))
        return i;
    }
  }
  return -1;
}
public void delete_book(String title)
{
  int i = book_index(title);
  if (i != -1) {
    book_list[i] = null;
    number_of_books--;
  }
}
```

Array applications (contd.)

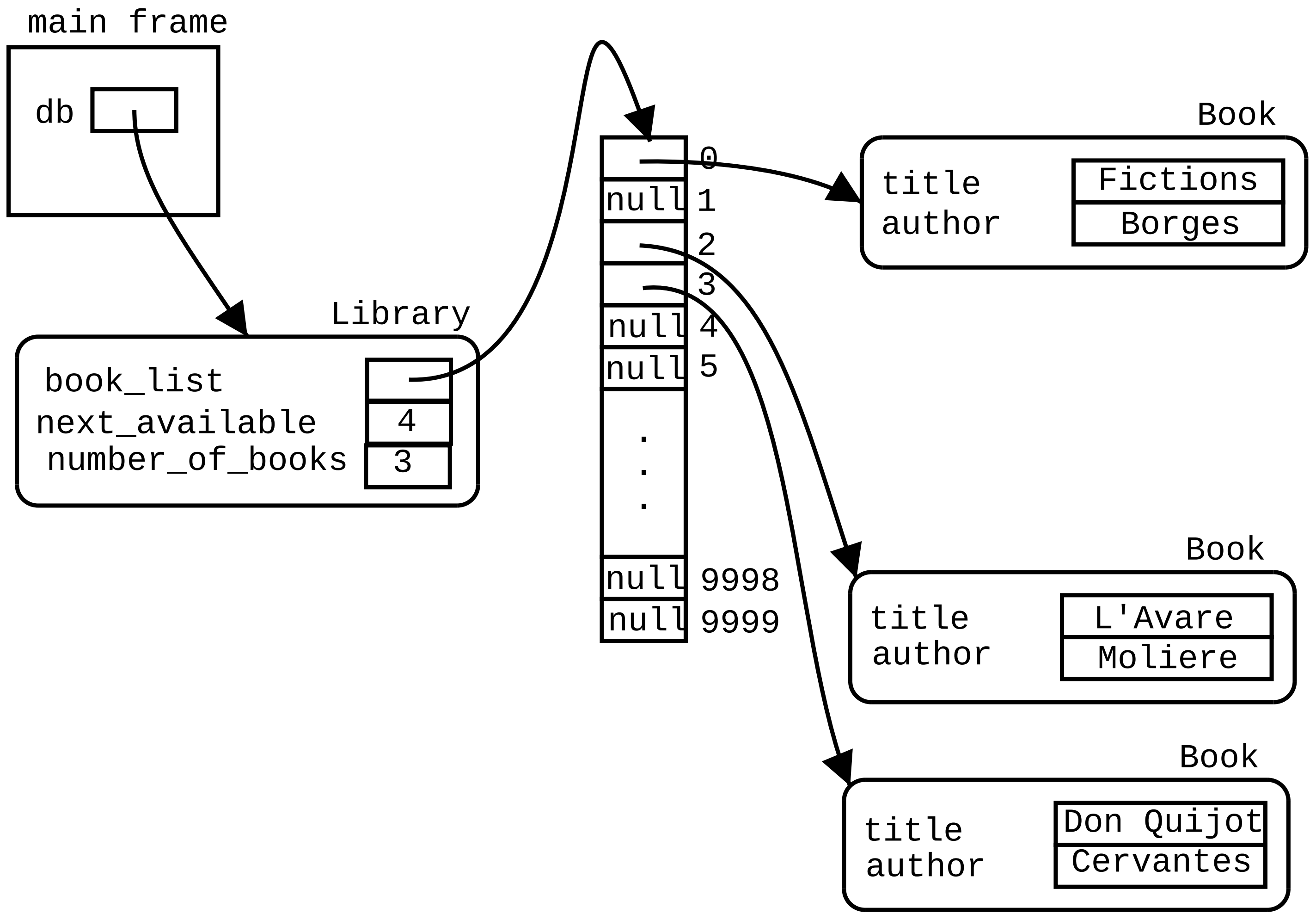
- But there's a problem: holes!

```
public class Test {
  public static void main(String[] args)
  {
    Library db = new Library(10000);
    Book m;
    m = new Book(``Fictions'',``Borges'');
    db.add_book(m);
    m = new Book(``Hamlet'',``Shakespeare'');
    db.add_book(m);
    m = new Book(``L\'Avare'',``Moliere'');
    db.add_book(m);
    db.delete_book(``Hamlet'');
    m = new Book(``Don Quijote'',``Cervantes'');
    db.add_book(m);
  }
}
```

Array applications (contd.)



Array applications (contd.)



Array applications (contd.)

- New algorithm for adding a book m:

1. Find an available slot i in book_list

2. Set book_list[i] to the book m

Array applications (contd.)

- Implementation

```
public void add_book(Book m)
{
  // Find an empty slot
  int index = 0;
  while (index < book_list.length
      && book_list[index] != null) {
    index++;
  }
  // Store the book
  if (index < book_list.length) {
    book_list[index] = m;
    number_of_books++;
  }
}
```

Array applications (contd.)

```
class Library {
  private Book[] book_list;
  public int number_of_books;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    number_of_books = 0;
  }

  public void add_book(Book m)
  {
    int index = 0;
    while (index < book_list.length
        && book_list[index] != null) {
      index++;
    }
    if (index < book_list.length) {
      book_list[index] = m;
      number_of_books++;
    }
  }
```

Array applications (contd.)

```
public int book_index(String title)
{
  for (int i=0; i < book_list.length; i++) {
    Book m = book_list[i];
    if (m != null && m.title().equals(title)) {
      return i;
    }
  }
  return -1;
}

public void delete_book(String title)
{
  int i = book_index(title);
  if (i != -1) {
    book_list[i] = null;
    number_of_books--;
  }
}
```

Array applications (contd.)

```
  public Book find_book(String title)
  {
    int i = book_index(title);
    if (i != -1) return book_list[i];
    return null;
  }
} // End of Library
```

Optimized Book database

Idea: instead of looking for an available cell each time we add a book, modify the delete method so that when we delete a book, move the last book of the list to the cell which just openned. This way, the array is not fragmented. This is, there are no holes, and all books are all grouped toghether at the beginning of the array.

```
public class Library {
  private Book[] book_list;
  private int next_available;

  public Library(int max_capacity)
  {
    book_list = new Book[max_capacity];
    next_available = 0;
  }
  // Continues below...
```

Optimized Book database

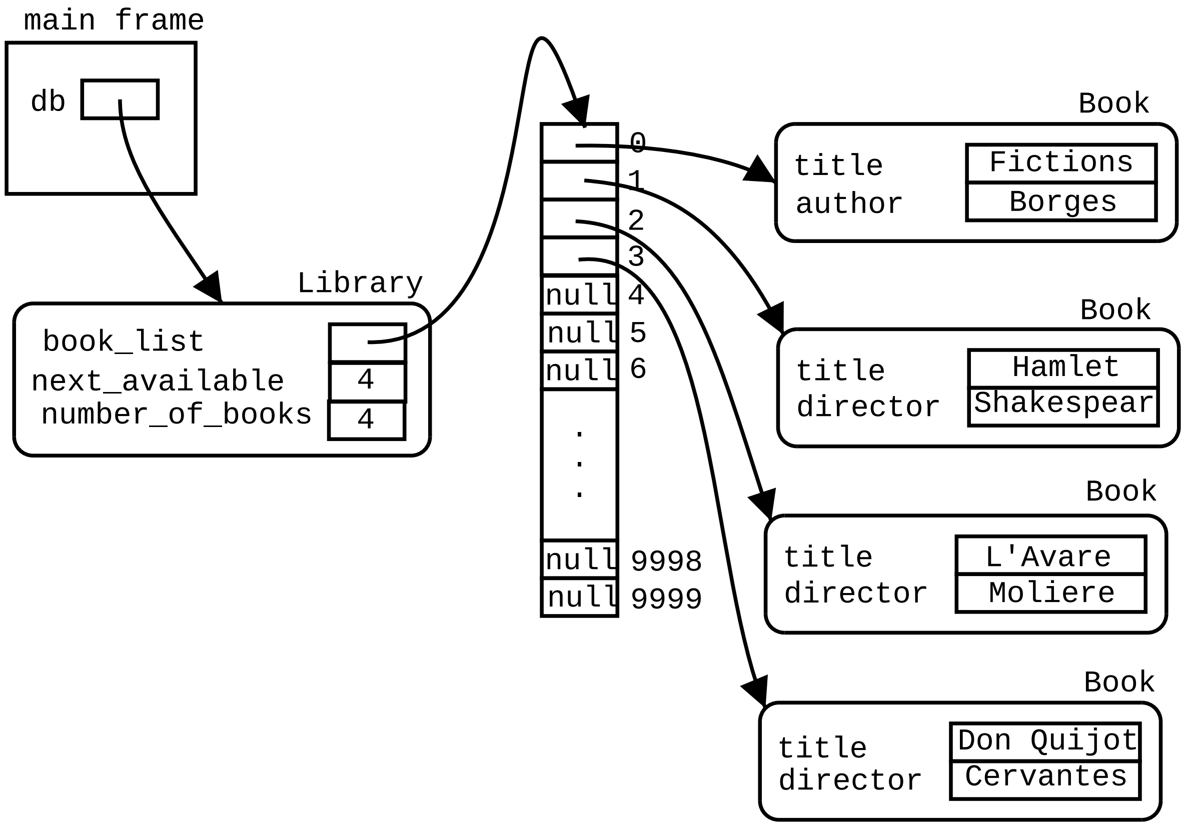
```
public void add_book(Book m)
{
  if (next_available < book_list.length) {
    book_list[next_available] = m;
    next_available++;
  }
}

public int book_index(String title)
{
  for (int i=0; i < book_list.length; i++) {
    Book m = book_list[i];
    if (m != null && m.title().equals(title)) {
      return i;
    }
  }
  return -1;
}
```

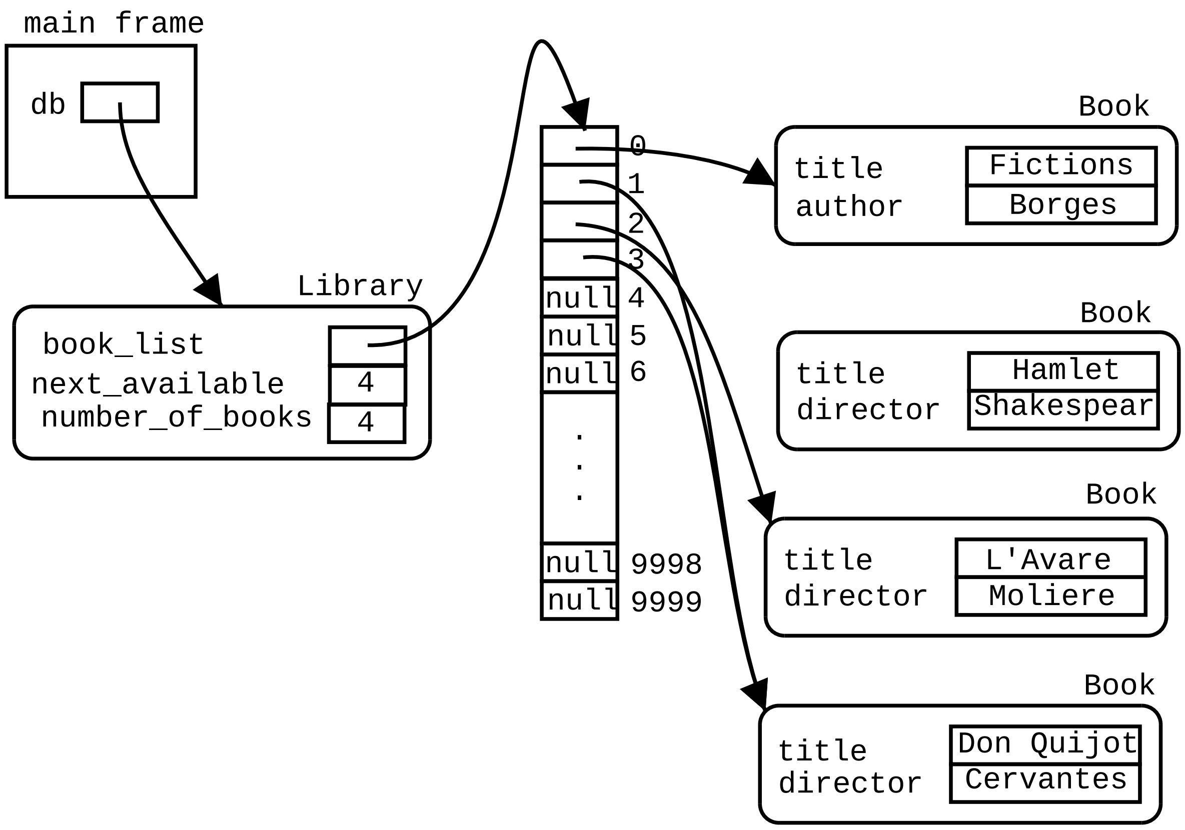
Optimized Book database

```
  public void delete_book(String title)
  {
    int i = book_index(title);
    if (i != -1) {
      book_list[i]=book_list[next_available-1];
      book_list[next_available - 1] = null;
      next_available--;
    }
  }
  public Book find_book(String t)
  {
    int i = book_index(t);
    if (i != -1) return book_list[i];
    return null;
  }
  public int number_of_books()
  {
    return next_available;
  }
}
```

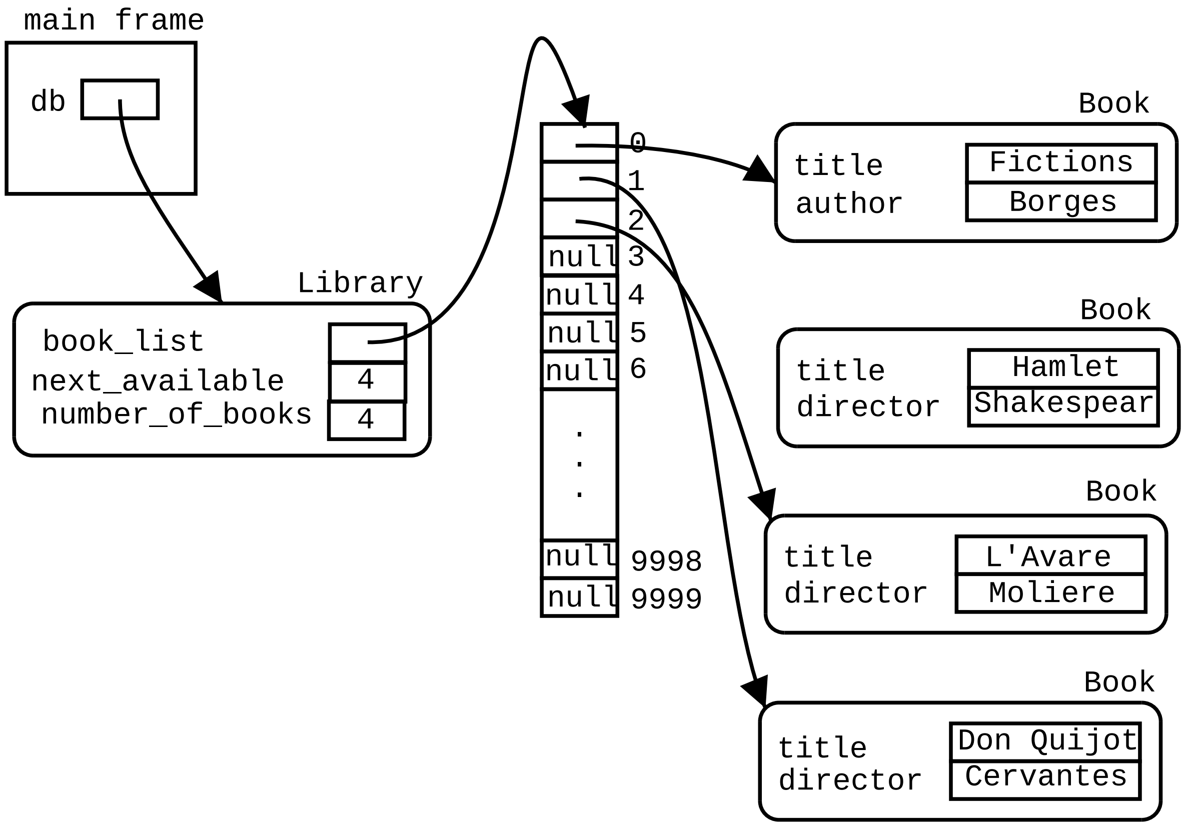
Optimized Book database



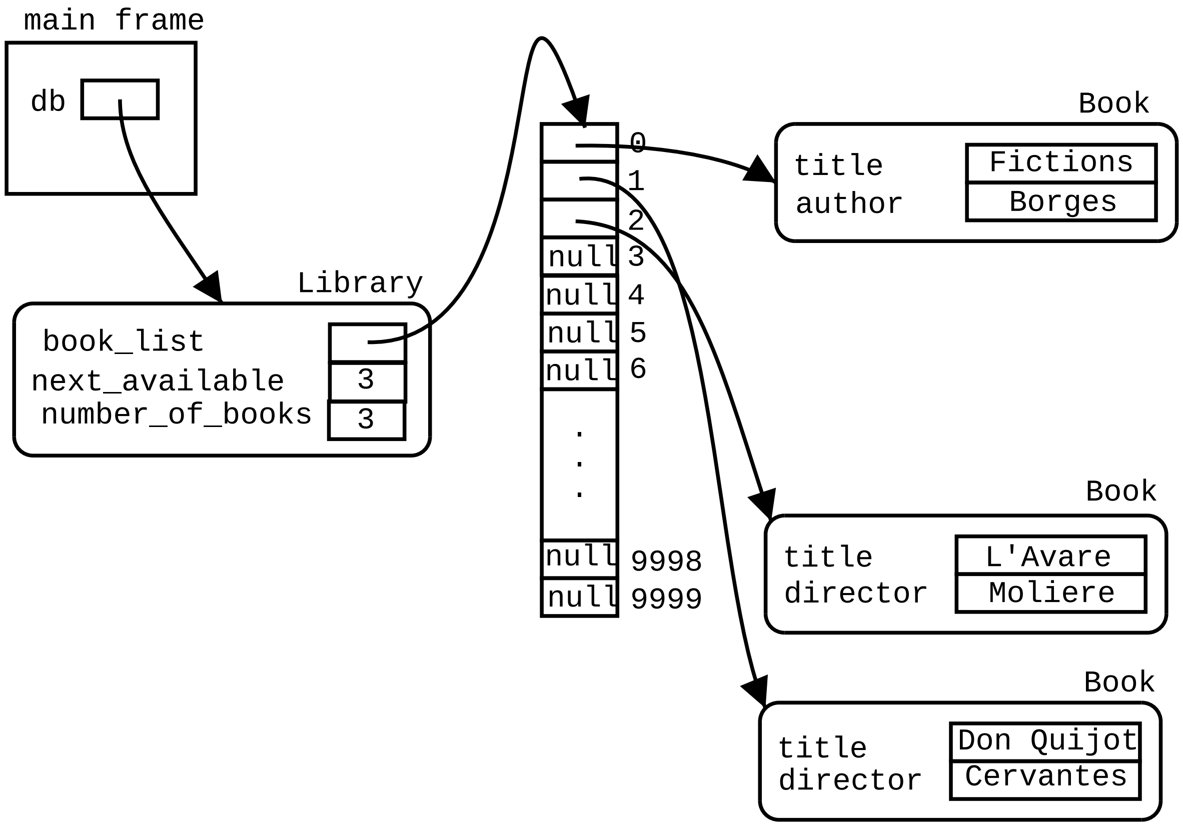
Optimized Book database



Optimized Book database



Optimized Book database



The end