Review

- 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*`;`

The null reference

- A variable whose type is a class is initialised to `null`.
- If a variable whose type is a class is not assigned an object (constructed with new,) and we try to access its attributes or methods, then a run-time error, called a "null-pointer exception" will occur.
- In the following example, if method `r` is called, a null pointer exception will occur:

```
class A { int x; }
class B {
  static void p(A u)
  {
    u.x = 7;   // Null pointer exception
  }
  static void r()
  {
    A v;  // v == null
    p(v);
  }
}
```

The null reference (contd.)

- We can avoid these errors by using an explicit check for a valid reference:

```
class A { int x; }
class B {
  static void p(A u)
  {
    if (u != null)
      u.x = 7;
  }
  static void r()
  {
    A v;  // v == null
    p(v);
  }
}
```

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;
        }
      }
    }
  }
```

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:

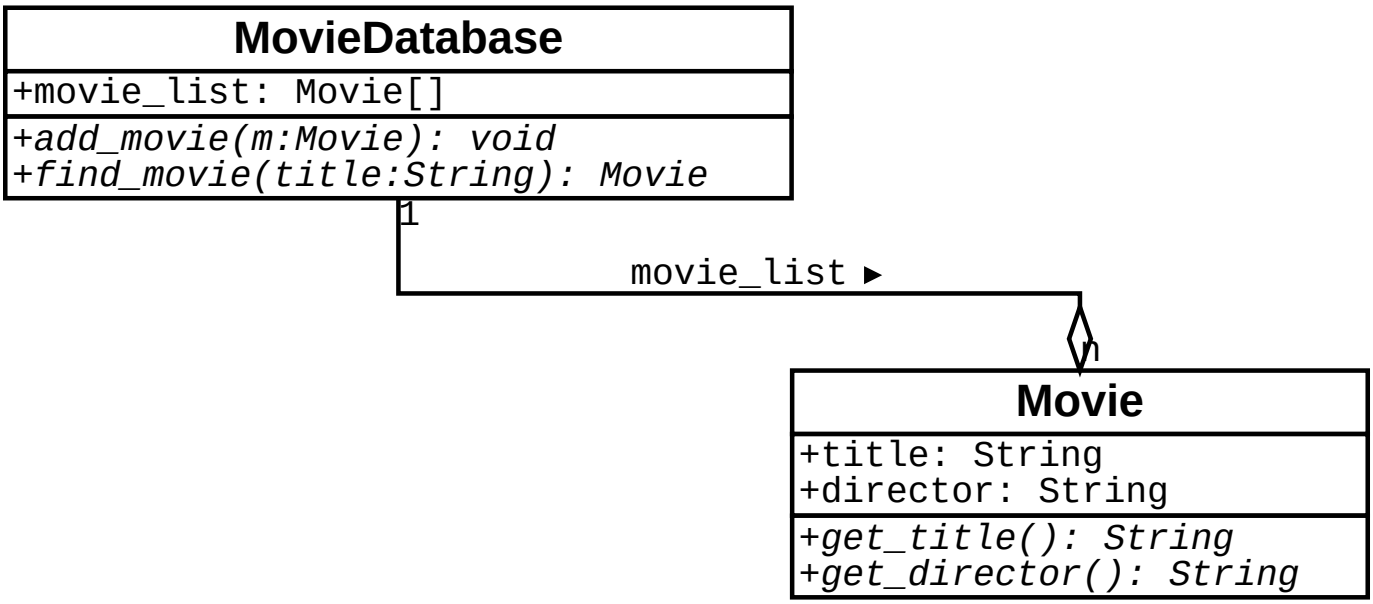
```
int[] a = { 1, 1, 2, 3, 5 };
Z[] u = { new Z(), new Z() };
```

Array applications

- Movie database
- Problem: Create a database of movies, which supports the operations of adding a new movie, and searching for a movie by title.
- Analysis:
  - Identify objects and classes:
    * Individual movies
    * A movie database
  - Relationships
    * Each movie *has a* title and a director
    * A movie database *has a* list of movies
  - Operations/Interactions/Behaviour
    * Adding movies to a database
    * Searching for a movie in a database

Array applications (contd.)

- Design
  - Class diagram



Array applications (contd.)

- Design (contd.)
  - Adding a movie m:

1. Set movie_list[i] to m, where i is the next available slot

So we need to remember the next available slot. We can do this by adding a new attribute to the database which is the index of the next available cell. Hence the add movie operation should now be:

1. If i < length of movie_list: (where i is the next available slot)
   (a) Set movie_list[i] to m,
   (b) increase i by 1

Array applications (contd.)

- Design (contd.)
  - Finding a movie with title t:

1. For each element movie_list[i] which is not null and such that i<the length of the list:
   (a) If the title ot movie_list[i] is equal to t then
      i. return movie_list[i]
2. If not found, return null

Array applications (contd.)

```
class Movie {
  private String title, director;
  public Movie(String t, String d)
  {
    title = t;
    director = d;
  }
  public String title() { return title; }
  public String director() { return director; }
  public Movie clone()
  {
    return new Movie(this.title, this.director);
  }
}
```

Array applications (contd.)

```
class MovieDatabase {
  private Movie[] movie_list;
  private int next_available;
  public int number_of_movies;

  public MovieDatabase(int max_capacity)
  {
    movie_list = new Movie[max_capacity];
    next_available = 0;
    number_of_movies = 0;
  }

  // Continues below...
```

```
  public void add_movie(Movie m)
  {
    if (next_available < movie_list.length) {
      movie_list[next_available] = m;
         // or m.clone();
      next_available++;
      number_of_movies++;
    }
  }
  public Movie find_movie(String title)
  {
    int index = 0;
    while (index < number_of_movies) {
      Movie m = movie_list[index];
      String t = m.title();
      if (t.equals(title)) {
        return m;
      }
      index++;
    }
    return null;
  }
} // End of MovieDatabase
```

Array applications (contd.)

- Second version of find

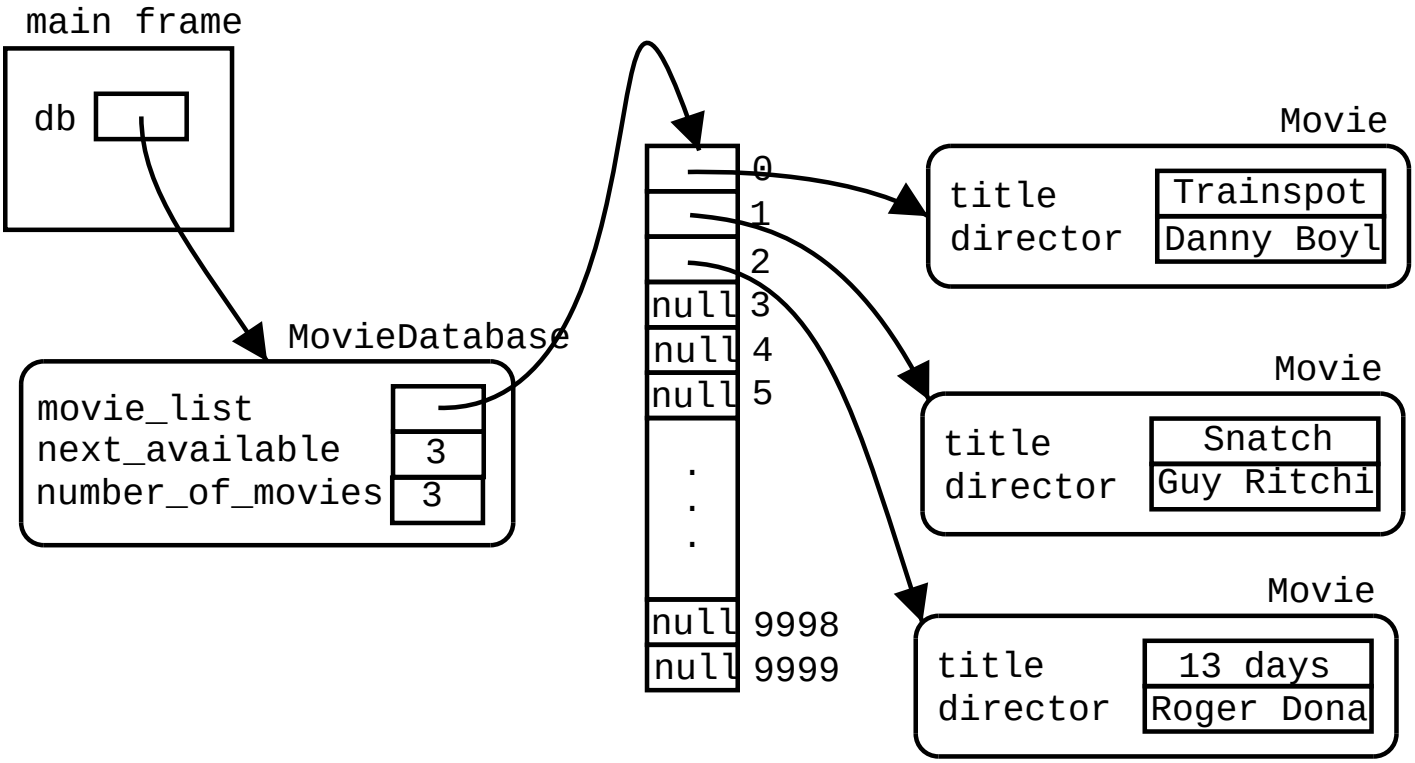
```
public Movie find_movie(String title)
{
  int index;
  index = 0;
  while (index < number_of_movies) {
    if (movie_list[index].title().equals(title)) {
      return movie_index[index];
    }
    index++;
  }
  return null;
}
```

Array applications (contd.)

- Using the database

```
public class Test {
  public static void main(String[] args)
  {
    MovieDatabase db = new MovieDatabase(10000);
    Movie m;
    m = new Movie(“Trainspotting”,“Danny Boyle”);
    db.add_movie(m);
    m = new Movie(“Snatch”,“Guy Ritchie”);
    db.add_movie(m);
    m = new Movie(“13 days”,“Roger Donaldson”);
    db.add_movie(m);
    Movie k = db.find_movie(“Snatch”);
    System.out.println(k.director());
  }
}
```

Array applications (contd.)



Array applications (contd.)

- Deleting elements from the database

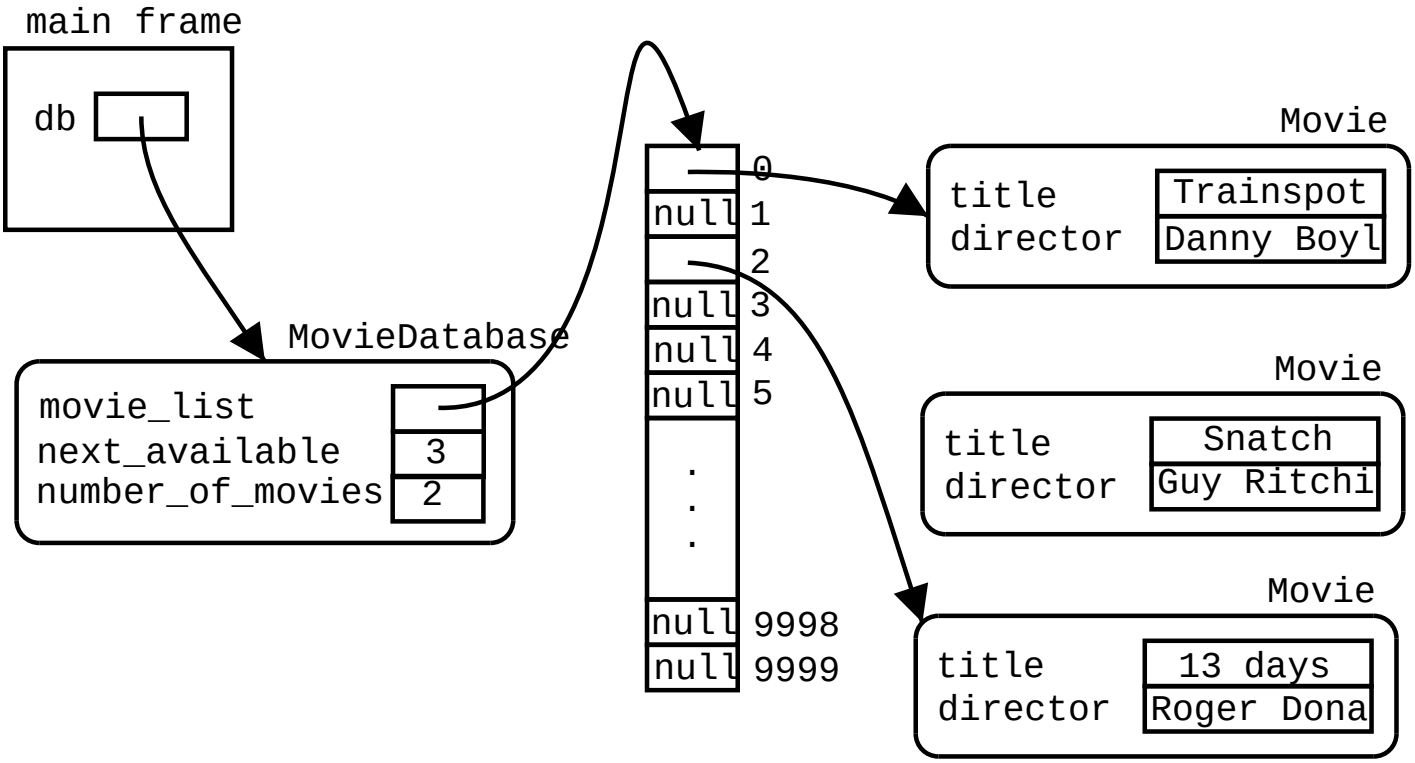
```
public int movie_index(String title)
{
  for (int i=0; i < movie_list.length; i++) {
    Movie m = movie_list[i];
    if (m != null && m.title().equals(title)) {
      return i;
    }
  }
  return -1;
}
public void delete_movie(String title)
{
  int i = movie_index(title);
  if (i != -1) {
    movie_list[i] = null;
    number_of_movies--;
  }
}
```

Array applications (contd.)

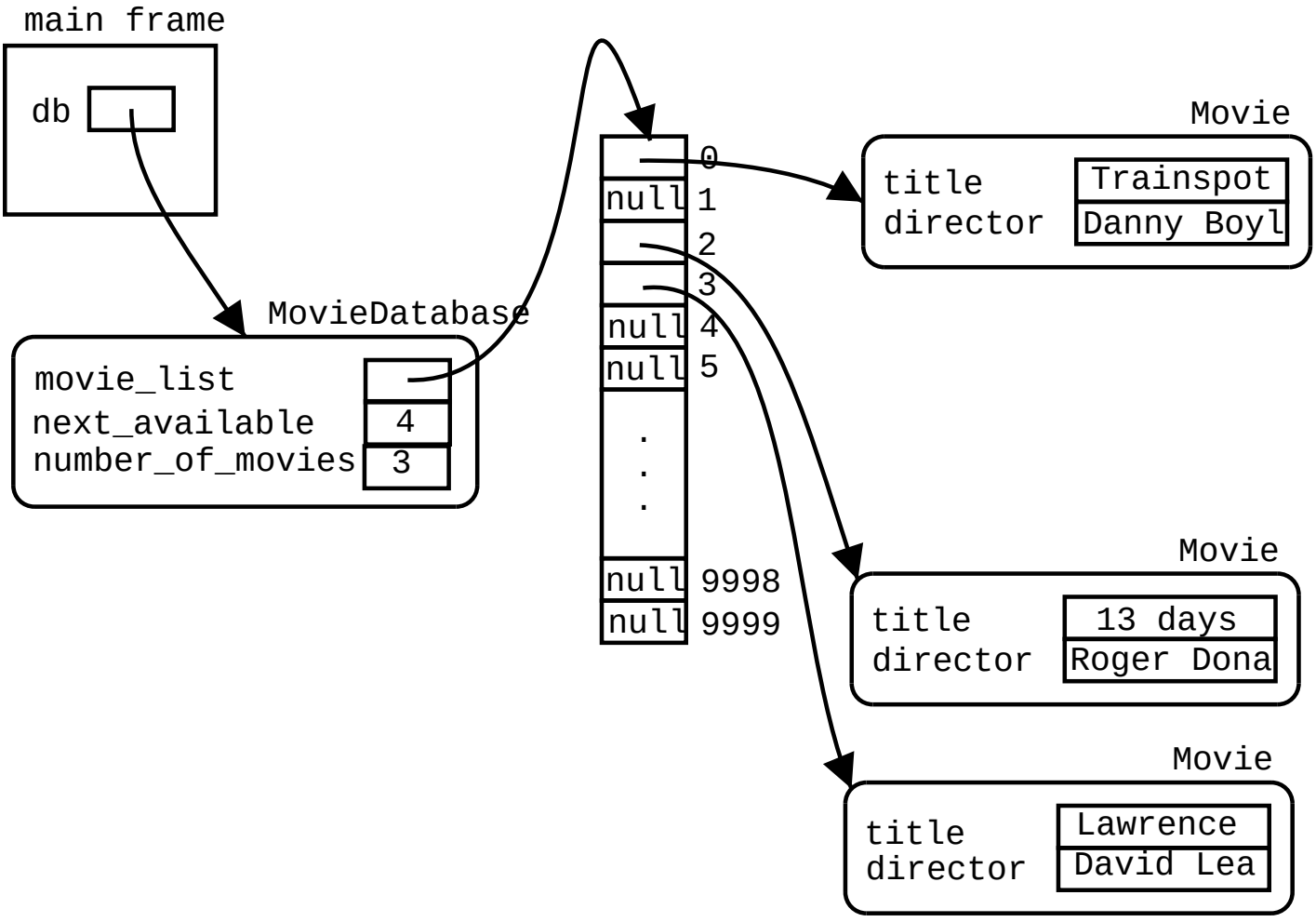
- But there's a problem: holes!

```
public class Test {
  public static void main(String[] args)
  {
    MovieDatabase db = new MovieDatabase(10000);
    Movie m;
    m = new Movie(“Trainspotting”,“Danny Boyle”);
    db.add_movie(m);
    m = new Movie(“Snatch”,“Guy Ritchie”);
    db.add_movie(m);
    m = new Movie(“13 days”,“Roger Donaldson”);
    db.add_movie(m);
    db.delete_movie(“Snatch”);
    m = new Movie(“Lawrence of Arabia”,“David Lean
    db.add_movie(m);
  }
}
```

Array applications (contd.)



Array applications (contd.)



Array applications (contd.)

- New algorithm for adding a movie m:

1. Find an available slot i in movie_list

2. Set movie_list[i] to the movie m

Array applications (contd.)

- Implementation

```
public void add_movie(Movie m)
{
  // Find an empty slot
  int index = 0;
  while (index < movie_list.length
      && movie_list[index] != null) {
    index++;
  }
  // Store the movie
  if (index < movie_list.length) {
    movie_list[index] = m;
    number_of_movies++;
  }
}
```

Array applications (contd.)

```
class MovieDatabase {
  private Movie[] movie_list;
  public int number_of_movies;

  public MovieDatabase(int max_capacity)
  {
    movie_list = new Movie[max_capacity];
    number_of_movies = 0;
  }

  public void add_movie(Movie m)
  {
    int index = 0;
    while (index < movie_list.length
        && movie_list[index] != null) {
      index++;
    }
    if (index < movie_list.length) {
      movie_list[index] = m;
      number_of_movies++;
    }
  }
```

Array applications (contd.)

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

public void delete_movie(String title)
{
  int i = movie_index(title);
  if (i != -1) {
    movie_list[i] = null;
    number_of_movies--;
  }
}
```

Array applications (contd.)

```
  public Movie find_movie(String title)
  {
    int i = movie_index(title);
    if (i != -1) return movie_list[i];
    return null;
  }
} // End of MovieDatabase
```

Optimized Movie database

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

```
public class MovieDatabase {
  private Movie[] movie_list;
  private int next_available;

  public MovieDatabase(int max_capacity)
  {
    movie_list = new Movie[max_capacity];
    next_available = 0;
  }
  // Continues below...
```

Optimized Movie database

```
public void add_movie(Movie m)
{
  if (next_available < movie_list.length) {
    movie_list[next_available] = m;
    next_available++;
  }
}

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

Optimized Movie database

```
  public void delete_movie(String title)
  {
    int i = movie_index(title);
    if (i != -1) {
      movie_list[i]=movie_list[next_available-1];
      movie_list[next_available - 1] = null;
      next_available--;
    }
  }
  public Movie find_movie(String t)
  {
    int i = movie_index(t);
    if (i != -1) return movie_list[i];
    return null;
  }
  public int number_of_movies()
  {
    return next_available;
  }
}
```