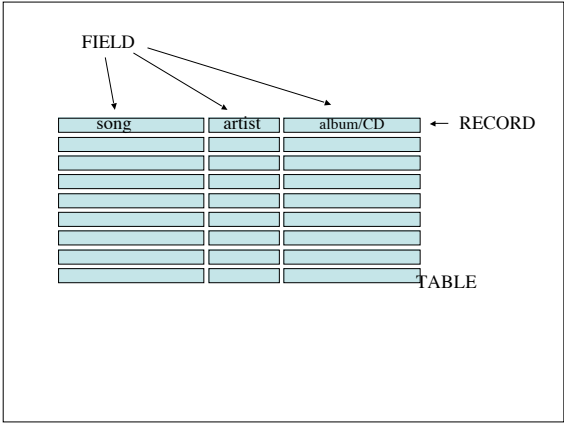
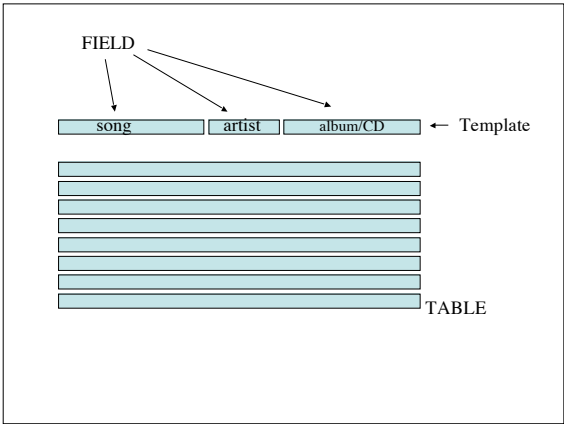


Tables and Maps
and the List interface
our first ADT (Abstract Data Type)





a musicRecord object

```
public class musicRecord
{
private String Song;
private String Album;
private String Artist;
private int Time;
private int TrackNumber;
//... all get and set methods
```

```
public String getSong()
{
return Song;
}

public String getArtist()
{
return Artist;
}
} // end class
```

a musicDatabase

```
// an array of musicRecords

musicRecord musicDatabase[ ] =
new musicRecord[ numberOfSongs ];
```

create a TreeMap, instead of an array

```
TreeMap myTable = new TreeMap();
for (x=0; x<numberOfSongs; x++)
{
    myTable.put(
        musicDatabase[x].getSong(),
        musicDatabase[x] );
}
// (key field, record )
```

use the key field

```
musicRecord myRecord =
(musicRecord) myTable.get("EboLorama");

// note cast
```

The List Interface

- A List, is an array, of indefinite size
- It grows and shrinks as you add and remove elements
- Two types of lists:
 - ArrayList
 - LinkedList
- *import java.util.*;*

Two “types”?

- An ArrayList is a general purpose array, but with list properties
 - certain methods
 - grows and shrinks
- A LinkedList contains “mappings” stored with each element which virtually sort the array (without moving elements around).
 - remember placeInSongList and placeInArtistList?

Two lists:

index

0	AC / DC	AC / DC		1
1	Beatles	Beatles	0	4
2	Incubus	Incubus	4	3
3	Ozzy	Ozzy	2	5
4	Eagles	Eagles	1	2
5	Pink Floyd	Pink Floyd	3	

A general-purpose interface

```
List myList;
```

decide later in the code which kind of list

```
myList = new ArrayList();
```

or

```
myList = new LinkedList();
```

some methods

To find the current size of a list we use the *size* method.

To empty a list use the *clear* method.

add() and get()

- add elements with add()
- get elements with get() - casting required

```
List sample = new ArrayList();  
sample.add("Hello");  
sample.add(14);
```

```
> javac tryStuff.java  
cannot resolve symbol sample.add( int );
```

```
List sample = new ArrayList();
sample.add("Hello");
// sample is now committed
sample.add("another String");
```

index-based methods

- **add**(int index, Object element)
Inserts the specified element at the specified position in this list
- **set**(int index, Object element)
Replaces the element at the specified position in this list with the specified element.
- **get**(int index)
Returns the element at the specified position in this list

the iterator

- traverses the list using *next()* and *hasNext()* methods
 - all Lists have an iterator, just go get it
- ```
List myList;
myList = new ArrayList();
Iterator itr = myList.iterator();
while (itr.hasNext ())
{ ...
```

---

---

---

---

---

---

---

---

## the Linked List

- has INTERNAL previous and next fields
- you don't see them
- can't manipulate them
- allows internal speedup of searching and sorting

```
List yourList = new ArrayList();
List myList = new LinkedList();
```

myList is faster

---

---

---

---

---

---

---

---