The List Abstract Data Type

Computer Science 112
Boston University

Christine Papadakis-Kanaris

Representing a Sequence: Arrays vs. Linked Lists

- Sequence – an ordered collection of items (position matters)
- Can represent any sequence using an array *or* a linked list

Representing a Sequence: Arrays vs. Linked Lists

- Sequence – an ordered collection of items (position matters
- Can represent any sequence using an array *or* a linked list

	array	***linked list***
representation in memory	elements occupy consecutive memory locations	nodes can be at arbitrary locations in memory; the links connect the nodes together
advantages	• provide random access (access to any item in constant time) • no extra memory needed for links	• can grow to an arbitrary length • allocate nodes as needed • inserting or deleting does *not* require shifting items
disadvantages	• have to preallocate the memory needed for the maximum sequence size • inserting or deleting can require shifting items	• no random access (may need to traverse the list) • need extra memory for links

Representing a Sequence: Arrays vs. Linked Lists

- Sequence – an ordered collection of items (position matters
- Can represent any sequence using an array *or* a linked list

- Regardless of the representation, the operations that we would need to perform on our list are:
  - `get an item in the list`
  - `add an item to the list`
  - `remove an item from the list`
  - `determine the length of the list`
  - `test if the list is full`

Abstract Data Types

- An *abstract data type* (ADT) is a model of a data structure that specifies:
  - the characteristics of the collection of data
  - the operations that can be performed on the collection

- It's *abstract* because it doesn't specify *how* the ADT will be implemented.

- A given ADT can have multiple implementations.

The List ADT

- A list is a sequence in which items can be accessed, inserted, and removed *at any position in the sequence*.
- The operations supported by our List ADT:
  - `getItem(i)`: get the item at position i
  - `addItem(item, i)`: add the specified item at position i
  - `removeItem(i)`: remove the item at position i
  - `length()`: get the number of items in the list
  - `isFull()`: test if the list already has the maximum number of items
- Note that we *don't* specify *how* the list will be implemented.

The List ADT

How can we ensure that the class we write implements all the methods as we have specified?

- A list is a sequence in wh inserted, and removed a
- The operations supported by ou
    - `getItem(i)`: get the item at position i
    - `addItem(item, i)`: add the specified item at position i
    - `removeItem(i)`: remove the item at position i
    - `length()`: get the number of items in the list
    - `isFull()`: test if the list already has the maximum number of items
- Note that we *don't* specify *how* the list will be implemented.

Specifying an ADT Using an Interface

- In Java, we can use an interface to specify an ADT:

```
public interface List {
    Object getItem(int i);
    boolean addItem(Object item, int i);
    Object removeItem(int i);
    int length();
    boolean isFull();
}
```

- An interface specifies a set of methods.
  - includes only their headers
  - does *not* typically include the full method definitions
- Like a class, it must go in a file with an appropriate name.
  - in this case: `List.java`

Specifying an ADT Using an Interface

- In Java, we can use an interface to specify an ADT:

```
public interface List {
    Object getItem(int i);
    boolean addItem(Object item, int i);
    Object removeItem(int i);
    int length();
    boolean isFull();
}
```

- An interface specifies a set of methods.
  - includes only their headers
  - does *not* typically include the full method definitions
- Like a class, it must go in a file with an appropriate name.
  - in this case: `List.java`
- Methods specified in an interface *must* be public,
  so we don't need the keyword `public` in the headers.

Implementing an ADT Using a Class

- To implement an ADT, we define a class.

- We specify the corresponding interface in the class header:

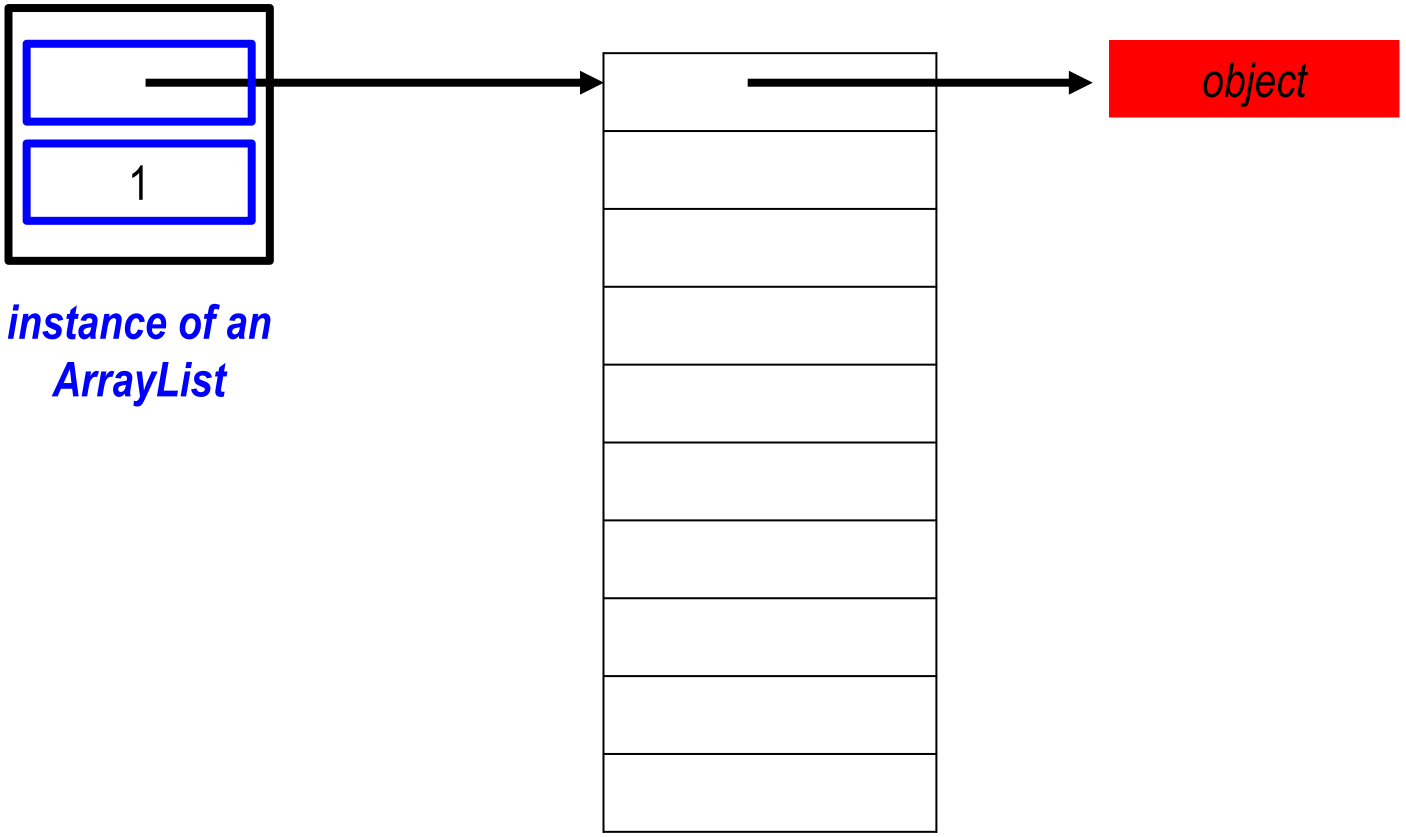
```
public class ArrayList implements List {
    ...
```

  - tells the compiler that the class will define *all* of the methods in the interface
  - if the class doesn't define them, it won't compile

- We'll look at two implementations of the `List` interface:
  - `ArrayList` – uses an array to store the items
  - `LLList` – uses a linked list to store the items

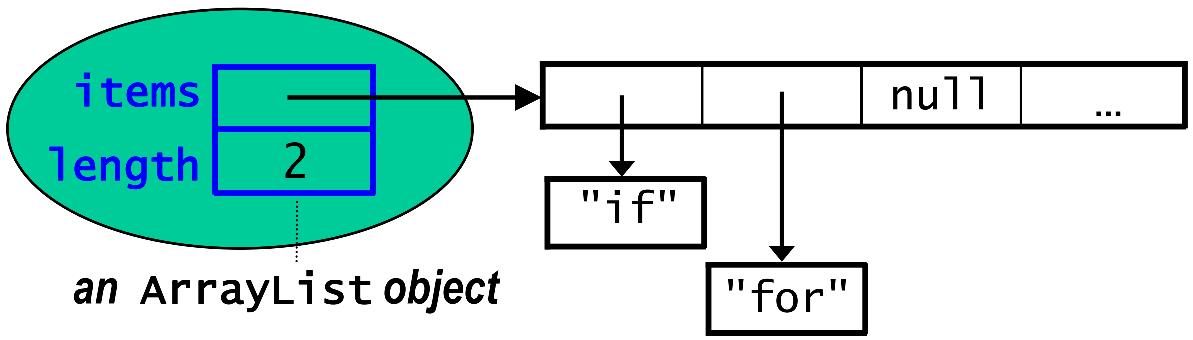
ArrayList Class

- Implementing the List interface with an **Array**



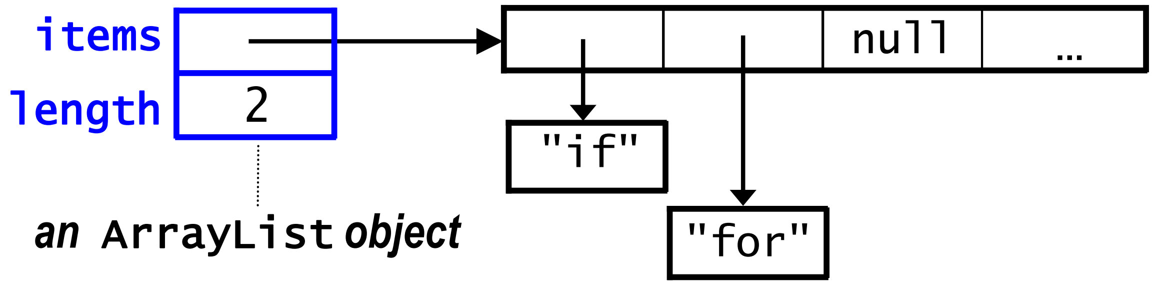
Implementing a List Using an Array

```
public class ArrayList implements List {
    private Object[] items;
    private int length;






    ...






}
```



Implementing a List Using an Array

```
public class ArrayList implements List {
    private Object[] items;
    private int length;

    public ArrayList(int maxSize) {
        this.items = new Object[maxSize];
        this.length = 0;
    }

    ...






}
```

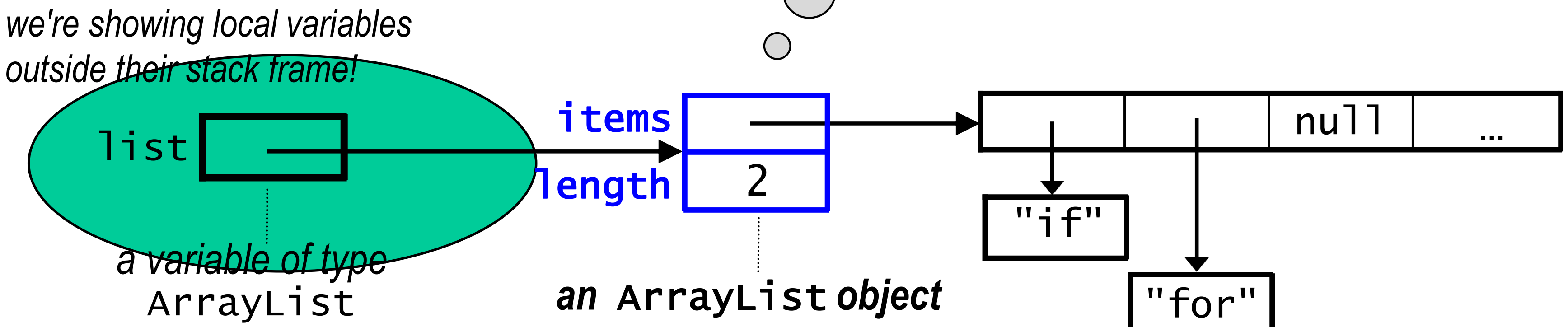


Implementing a List Using an Array

```
public class ArrayList implements List {
    private Object[] items;
    private int length;

    public ArrayList(int maxSize) {
        this.items
        this.len
    }

    ...

}
```

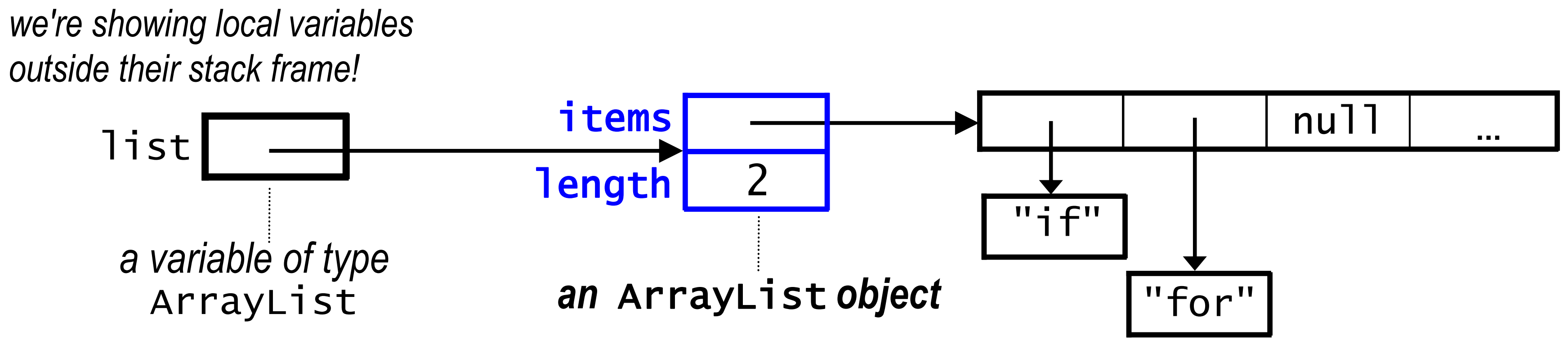
Example:

```
ArrayList list = new ArrayList(n);
…
 list.addItem("if", 0);
 list.addItem("for", 1);
```


we're showing local variables outside their stack frame!
list
a variable of type ArrayList
items
length
2
an ArrayList object
null
…
"if"
"for"

Implementing a List Using an Array

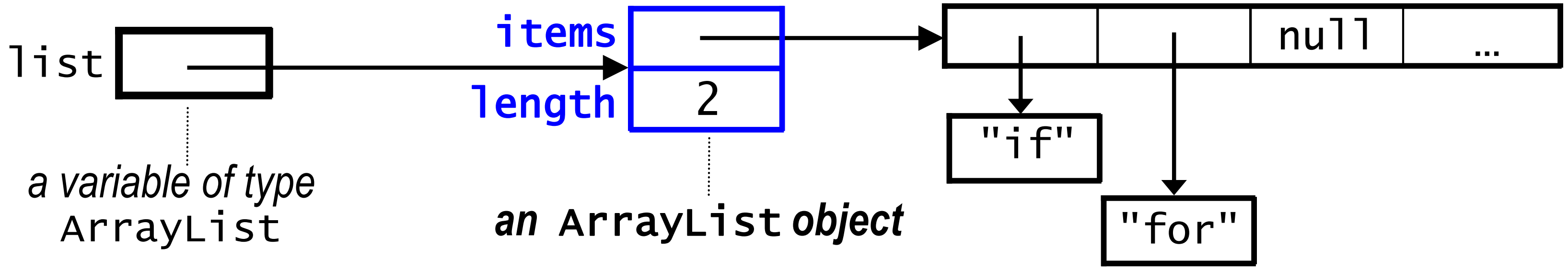
```
public class ArrayList implements List {
    private Object[] items;
    private int length;

    public ArrayList(int maxSize) {
        this.items = new Object[maxSize];
        this.length = 0;
    }

    public int length() {
        return this.length;
    }

    public boolean isFull() {
        return (this.length == this.items.length);
    }
    ...
}
```


we're showing local variables outside their stack frame!
list
items
length
2
null
...
"if"
"for"
a variable of type ArrayList
an ArrayList object

Implementing a List Using an Array

```
public class ArrayList implements List {
    private Object[] items;
    private int length;

    public ArrayList(int maxSize) {
        this.items = new Object[maxSize];
        this.length = 0;
    }

    public int length() {
        return this.length;
    }

    public boolean isFull() {
        return (this.length == this.items.length);
    }
    ...
}
```

we're showing local variables outside their stack frame!



More to follow next week….