public abstract int length();

Returns the number of elements in this ALlist.

...
... 
{
    return 0;
}

public int IntLength()
    returns 0 //
...

public class EmptvList extends ArrayList

Computing the Length of a Scheme-like List (cont.)
...  
{
    return 1 + -rest.length();
}

public int length()
    
    Returns 1 + the number of elements in -rest.  //

...  
}

public class NILIST extends ALIST

Computing the Length of a Scheme-like List (cont.)
An Implementation in Scheme
null is not an object. It cannot perform computations.

```java
public int length()
    returns 0; // An EmptyList object can perform a computation.
```

An EmptyList Object vs. null
is called the Singleton Design Pattern.

There is a way to design a class to ensure such uniqueness property. It

throughout the life of a program?

How can we ensure that only one instance of EmptyList can be created

set.

– This concept is akin to that of the empty set: there is only one empty

Conceptually, there is only one empty list in the „world“.

The Singleton Pattern
The Singleton Pattern (cont.)

- The following UML diagram describes the pattern:

  ```
  $\text{SingletonClass}$
  - $\text{instance: SingletonClass}$
  + $\text{getInstance() : SingletonClass}$

  Private constructor has empty body. No external client can call this private constructor for instantiation.

  Check if instance is null or not. If it is null, instantiate it.

  return instance;
  ```

Note: The field instance and the method getInstance() are of class scope (i.e. static).
(Recall that we saw the Factory Pattern in Lab 2.)

The class SingletonClass is appropriately called a "factory". In this very special case, SingletonClass manufactures its own (unique) instance, though unique, though unique, of the SingletonClass.

The method UniqueInstance() is called a "factory" method as it is used to manufacture an instance.
public static EmptyList makeEmptyList()
{
    if (instance == null)
    
    return instance;

    instance = new EmptyList();

    return instance;
}

private static EmptyList
{
}

private EmptyList()

    // EmptyList, Singleton, and every List uses it.
    // EmptyList is EmptyList. I.e., there is only true
    // Instance. And EmptyList extends ArrayList.
    // NOTE: The constructor is private so that no client can

    private static EmptyList instance;

    public class EmptyList extends ArrayList

    One EmptyList Object is Enough.
... {
)

private EmptyList()

    /*
     * Empty List, Singleton, and every List uses it.
     * Instantiate an EmptyList. I.e., there is one "true"
     * Note: The constructor is private so that no client can
     **/

public final static EmptyList Singleton = new EmptyList();

public class EmptyList extends ArrayList

One EmptyList Object is Enough
class/method/field cannot be both final and abstract. Thus, in some sense, final is the opposite of abstract.

from being extended or overridden.

- the field
- the method
- the class

The final modifier prevents
public final static EmptyList singleton = new EmptyList;

- Example
  initializer

• If a field is declared final, then its declaration must include a variable

The final Modifier (cont.)