Data Structures

Abstract Data Types

August 11, 2016

Outline

Outline

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

Examples

- integer = {0, +1, -1, +2, -2, +3, -3, ...}
- daysOfWeek = {S,M,T,W,Th,F,Sa}

What is a data structure?

Intuition

A data object is a set or collection of instances!!!

Examples

- integer = {0, +1, -1, +2, -2, +3, -3, ...}
- daysOfWeek = {S,M,T,W,Th,F,Sa}

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Thus

Data Structure $\approx$ Data object + relationships that exist among instances and elements that comprise an instance.

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Thus

Data Structure ≈ Data object + relationships that exist among instances and elements that comprise an instance.

Those instance may be related!!!

Something quite basic

Instances may or may not be related.

Example

myDataObject = {apple, chair, 2, 5.2, red, green, Jack}

Thus

Data Structure $\approx$ Data object + relationships that exist among instances and elements that comprise an instance.

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Examples

Among instances of integers

- $369 < 370$
- $280 + 4 = 284$

Thus

The relationships are usually specified by specifying operations on one or more instances.

Examples

add, subtract, predecessor, multiply

Outline

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

Abstract Data Type

Data Type

A data type such as **int** or **double** is a group of values and operations on those values that is defined within a specific programming language.

Abstract Data Type

An Abstract Data Type, or ADT, is a specification for a group of values and the operations on those values that is defined conceptually and independently of any programming language.

Thus

A data structure is an implementation of an ADT within a programming language.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

1. An abstract object may be passed as a parameter to a procedure.
2. An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

Using Abstract Data Types

Something Notable

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

Using Abstract Data Types

1. An abstract object may be operated upon by the operations which define its abstract type.
2. An abstract object may be passed as a parameter to a procedure.
3. An abstract object may be assigned to a variable.

For More

We have the following

"PROGRAMMING WITH ABSTRACT DATA TYPES" by Barbara Liskov and Stephen Zilles

Abstract Data Types: An Example

We will start with

An example using Lists.

Actually in Java

An interface in the Java programming language is an abstract type that is used to specify an interface (in the generic sense of the term) that classes must implement.

Abstract Data Types: An Example

We will start with

An example using Lists.

Actually in Java

An interface in the Java programming language is an abstract type that is used to specify an interface (in the generic sense of the term) that classes must implement.

Outline

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

Linear (or Ordered) Lists

Definition

A linear list is a data structure that holds a sequential list of elements.

Instances

They have the following structure, $(e_0, e_1, e_1, ..., e_{n-1})$

Properties

- Where e_i denotes a list element
- $n \geq 0$ is finite
- List size is n

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

What are the relations inside of a Liner List?

Having this

$$L = (e_0, e_1, e_2, ..., e_{n-1})$$

Relationships

- e_0 is the zero'th (or front) element
- e_{n-1} is the last element
- e_i immediately precedes e_{i+1}

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Examples

Simple ones

- Students in COP3530 = (Jack, Jill, Abe, Henry, Mary, ..., Judy)
- Exams in COP3530 = (exam1, exam2, exam3)
- Days of Week = (S, M, T, W, Th, F, Sa)
- Months = (Jan, Feb, Mar, Apr, ..., Nov, Dec)

Outline

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

Which operations must be supported?

We need the size of a linear list

Hey, we need to know how many elements the linear list has.

Determine list size for

$L = (a, b, c, d, e)$

So, we need to have a operation that returns the size

- size(L)=5

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: get(theIndex)

Get operations

Get element with given index.

For example

$L = (a, b, c, d, e)$

Thus

- get(0) = a
- get(2) = c
- get(4) = e
- get(-1) = error
- get(9) = error

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) = 2
- indexOf(a) = 0
- indexOf(z) = -1

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) = 2
- indexOf(a) = 0
- indexOf(z) = -1

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) = 2
- indexOf(a) = 0
- indexOf(z) = -1

Linear List Operations: indexOf(theElement)

IndexOf operations

Determine the index of an element.

Example

$L = (a, b, d, b, a)$

Thus

- indexOf(d) = 2
- indexOf(a) = 0
- indexOf(z) = -1

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

Thus

- **add(0,h)** $\Longrightarrow L = (h, a, b, c, d, e, f, g)$
  - Thus index of a, b, c, d, e, f, g increased by 1

Linear List Operations: add(theIndex, theElement)

Definition

Add an element so that the new element has a specified index.

If we have

$L = (a, b, c, d, e, f, g)$

Thus

- **add(0,h)** $\Longrightarrow L = (h, a, b, c, d, e, f, g)$
  - Thus, index of a, b, c, d, e, f, g increases by 1.

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- add(10,h) $\Longrightarrow$ error
- add(-6,h) $\Longrightarrow$ error

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- **add(10,h)** $\Longrightarrow$ error
- add(-6,h) $\Longrightarrow$ error

Example

add(2,h) using the original L

- $\Longrightarrow L = (a, b, h, c, d, e, f, g)$
  - index of c, d, e, f, g increases by 1

What about out of the bound

- **add(10,h)** $\Longrightarrow$ error
- **add(-6,h)** $\Longrightarrow$ error

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

Example

- **Remove(2)** returns c and L becomes (a, b, d, e, f, g)
  - Index of d, e, f, g decreases by 1

Linear List Operations: remove(theIndex)

Definition

Remove and return element with given index.

For example

$L = (a, b, c, d, e, f, g)$

Example

- **Remove(2)** returns c and L becomes (a, b, d, e, f, g)
  - Index of d, e, f, g decreases by 1

What about the Error

For

$L = (a, b, c, d, e, f, g)$

Thus

- remove(-1) $\Longrightarrow$ error
- remove(20) $\Longrightarrow$ error

What about the Error

For

$L = (a, b, c, d, e, f, g)$

Thus

- remove(-1) $\Longrightarrow$ error
- remove(20) $\Longrightarrow$ error

The Final Abstract Data Type for Linear List

Linear List

AbstractDataType LinearList
{
  instances
    ordered finite collections of zero or more elements
  operations
    isEmpty(): return true iff the list is empty, false otherwise
    size(): return the list size (i.e., number of elements in the list)
    get(index): return the element with "index" index
    indexOf(x): return the index of the first occurrence of x in the list, return -1
                if x is not in the list
    remove(index): remove and return the indexth element, elements with higher
                   index have their index reduced by 1
    add(theIndex, x): insert x as the index of th element, elements with
                      theIndex $\geq$ index have their index increased by 1
    output(): output the list elements from left to right
}

Java Interface

We can implement this idea of Abstract Data Types

- Using Java through the generic interface!!!
- An interface may include constants and abstract methods (i.e., methods for which no implementation is provided).

Example for List

Java Interface

We can implement this idea of Abstract Data Types

- Using Java through the generic interface!!!
- An interface may include constants and abstract methods (i.e., methods for which no implementation is provided).

Example for List

```
public interface LinearList<Item>{
    public boolean isEmpty();
    public int size();
    public Item get(int index);
    public int  indexOf(Item myobject);
    public void add(int index,Item myobject);
    public Item remove(int index);
    public String toString();
}
```

Implementing the Java Interface

Using a calss to implement the interface

```
class SimpleArrayList <Item> implements
                                LinearList <Item>{

// code for all methods must be provided

}
```

Outline

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes

- An abstract class is a special kind of class that cannot be instantiated.
- It is a contract that is used to define hierarchies for all subclasses.

- An interface has no implementation.
- It only has the definition or the signature of the methods.

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes

- An abstract class is a special kind of class that cannot be instantiated.
- It is a contract that is used to define hierarchies for all subclasses.

Interface

- An interface is not a class.
- An interface has no implementation.
- It only has the definition of the methods without the body.

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes

- An abstract class is a special kind of class that cannot be instantiated.
- It is a contract that is used to define hierarchies for all subclasses.

Interface

- An interface is not a class.
- An interface has no implementation.
- It only has the definition of the methods without the body.

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes

- An abstract class is a special kind of class that cannot be instantiated.
- It is a contract that is used to define hierarchies for all subclasses.

Interface

- An interface is not a class.
- An interface has no implementation.
- It only has the definition of the methods without the body.

Abstract Classes Vs Interfaces

However, we have another way to implement ADT

Abstract Classes

Abstract Classes

- An abstract class is a special kind of class that cannot be instantiated.
- It is a contract that is used to define hierarchies for all subclasses.

Interface

- An interface is not a class.
- An interface has no implementation.
- It only has the definition of the methods without the body.

Important

We have

A Java class may implement as many interfaces as it wants but can extend at most one class.

This is why

Once you have chosen your design you can enforce the structure using Abstract Classes!!!

Important

We have

A Java class may implement as many interfaces as it wants but can extend at most one class.

This is why

Once you have chosen your design you can enforce the structure using Abstract Classes!!!

Example of Abstract Class

Code

```
public abstract class LinearList<Item>{
    public boolean isEmpty();
    public int size();
    public Item get(int index);
    public int  indexOf(Item myobject);
    public void add(int index,Item myobject);
    public Item remove(int index);
    public String toString();
}
```