Abstract Data Types

An abstract data type (ADT) is a data abstraction where the implementation is hidden behind a set of operations.

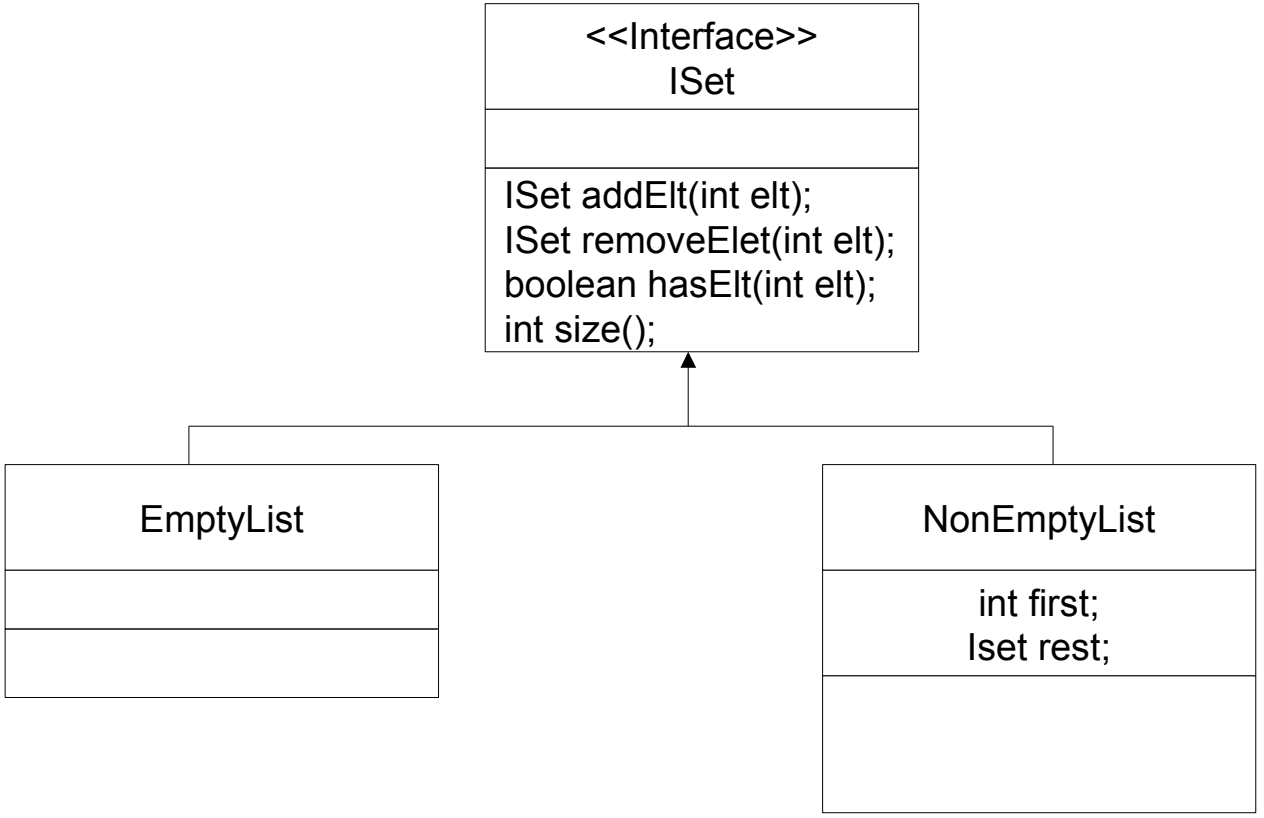
The operations are precisely specified independent of any particular implementation.

Sets

- A set is a collection of elements such that
  - There are no duplicate elements
  - There is no order
- Operations
  - addElt: set element → set
  - removeElt: set element → set
  - hasElt: set element → boolean
  - size: set → int

```
interface Iset {
      ISet addElt(int elt);
      ISet removeElet(int elt);
      boolean hasElt(int elt);
      int size();
}
```

```
;a ListOfString is one of
;   empty
;   (cons String ListOfString)
```


<<Interface>>
ISet
ISet addElt(int elt);
ISet removeElet(int elt);
boolean hasElt(int elt);
int size();
EmptyList
NonEmptyList
int first;
Iset rest;

Two Ways of Thinking About Sets

- Option A: There may be duplicates, but trust that the user will never know about them
- Option B: Trust that there are no duplicates in the list

How to Define addElt()?

- Option A: add the element to the front of the list (just like cons does in Racket)
- Option B: Check if the element is already in the list.
  - If it is, don’t add it again.
  - If it isn’t, add the element to the front of the list.

What Would This Look Like?

```
ISet s = new EmptyList();
...s.addElt(12).addElt(4).addElt(12).
addElt(12).addElt(9)
```

Option A: 9 12 12 4 12
Option B: 9 4 12

Option A		Option B
Add elt to front	addElt	Add to front only if not already in list
Have to remove all occurrences of the element	removeElt	Find the first (and only) occurrence and remove it
Search the list and return true at the first occurrence of the element, or return false if you look at all elements and you don’t find the element you want	hasElt	Same as option A
Count the unique elements in the list	size	Count all element

Big-O

	Option A	Option B
addElt	O(1)	O(n)
removeElt	O(n)	O(n)
hasElt	O(n)	O(n)
size	$O(n^2)$	O(n)