Checking it twice

Lists

CS111 Computer Programming

Department of Computer Science
Wellesley College

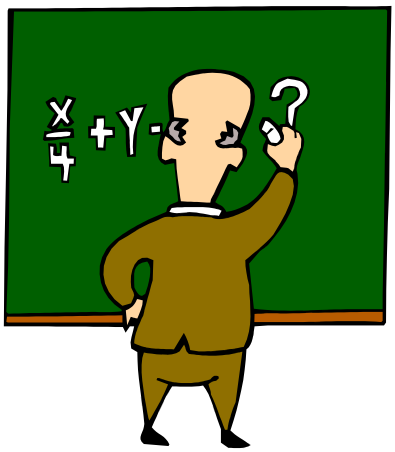
IntList represents lists of integers

- Methods in the (Java defined) Math class are static; you invoke them on the Math class:

Class name → `Math`; Class method → `abs(y)`

```
int x = Math.abs(y);
```

- Similarly all IntList methods are class methods. You invoke them on the IntList class.*

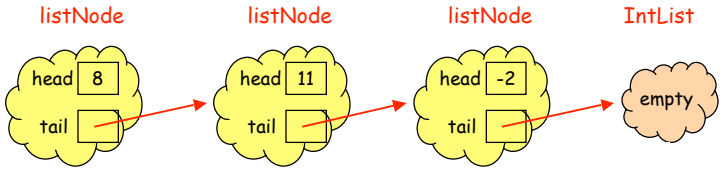


*Not an object.

Intlists are defined recursively

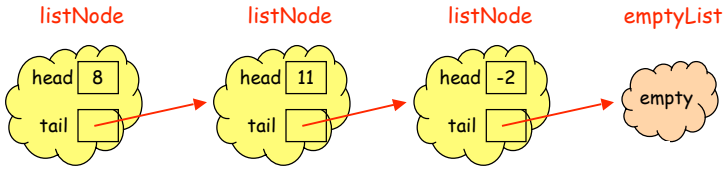
An IntList is either

- the empty list, or
- a (non-empty) listNode with two parts:
  1. a head which is an integer, and
  2. a tail which is another IntList.



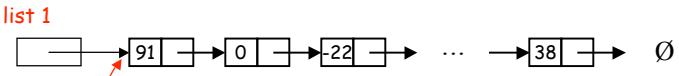
Box-and-pointer notation

We represent listNode and emptyList objects using a shorthand known as box-and-pointer notation.

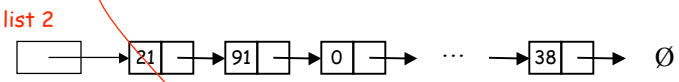


Why are lists useful?

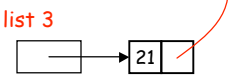
Lists are good for storing many numbers (but you don't know how many in advance).



Lists are also very flexible. E.g., here is a new list which starts with 21, and then the rest are the same as before.



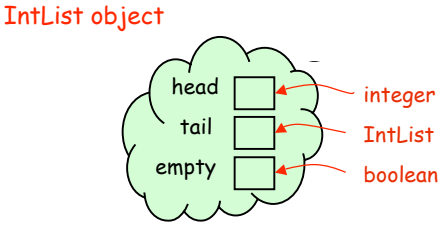
We could create list 2 from scratch, or we could use the first list as the tail of a new list.



Exploring IntList

IntList has two constructor methods

```
public IntList()                          // creates empty list
public IntList(int head, IntList tail) // creates a list with
                                          // given head and tail.
```



Five core IntList methods*

`public static int head (IntList L)`
Returns the integer that is the head component of the integer list node **`L`**.

`public static IntList tail (IntList L)`
Returns the integer list that is the tail component of the integer list node **`L`**.

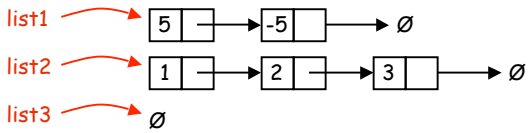
`public static boolean isEmpty (IntList L)`
Returns **true** if **`L`** is an empty integer list and **false** if **`L`** is an integer list node.

`public static IntList empty()`
Returns an empty integer list.

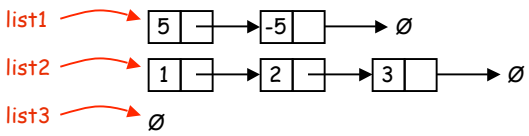
`public static IntList prepend (int n, IntList L)`
Returns a new integer list node whose head is **`n`** and whose tail is **`L`**.

*See the full contract for more.

Our working example



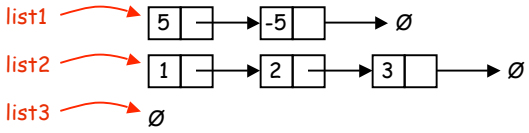
isEmpty method




```
IntList.isEmpty(list1);
IntList.isEmpty(list2);
IntList.isEmpty(list3);
```

`public static boolean` **`isEmpty (IntList L)`**
Returns **true** if `L` is an empty integer list and **false** if `L` is an integer list node.

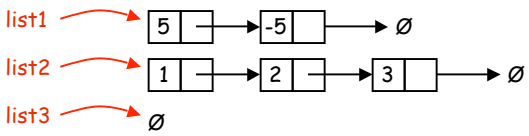
head method




```
IntList.head(list1);
IntList.head(list2);
IntList.head(list3);
```

`public static int` **`head (IntList L)`**
Returns the integer that is the head component of the integer list node `L`.

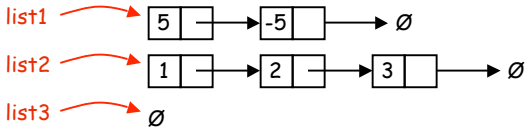
tail method




```
IntList.tail(list1);
IntList.tail(list2);
IntList.tail(list3);
```

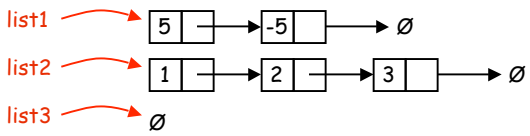
`public static IntList tail (IntList L)`
Returns the integer list that is the tail component of the integer list node `L`.

heads and tails




```
IntList.head(IntList.tail(list2))
IntList.head(IntList.tail(IntList.tail(list2)))
```

heads and tails




```
IntList.prepend(3, IntList.empty())
IntList.prepend(2, IntList.prepend(3, IntList.empty()))
IntList.prepend(1, IntList.prepend(2,
                                   IntList.prepend(3, IntList.empty())))
```

```
public static IntList prepend (int n, IntList L)
```
Returns a new integer list node whose head is **n** and whose tail is **L**.

Recursive list methods

```
// Returns the integer sum of all element in L
public static int sumList (IntList L)
{




}
```

Recursive list methods

```
// Returns the number of element in list L
public static int length (IntList L)
{




}
```


Recursive list methods

```
// Returns the max number in list L
public static int maxList (IntList L)
{




}
```

Recursive list methods

```
// Returns true if all the elements in L are positive
// and returns false otherwise
public static boolean areAllPositive (IntList L)
{




}
```


Filter methods

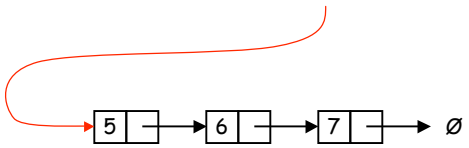
A *filter* method passes some elements through and holds other elements back.

More formally: A *filter* method transforms one list to another by keeping only those elements satisfying a given predicate.

Filtering divisors

- Method filterDivisors(int n, IntList list) returns a list containing elements of "list" that divide "n" evenly.
- For example,

```
filterDivisors(30, list)
```



filterDivisors(int n, IntList list);

```
// Returns sublist containing divisors of n
public static intList filterDivisors(int n, IntList list)
{
   if (               )  {          // base case
        return
   } else if (                ) {  // recursive case

   } else {

  }
}
```

filterDivisors(int n, IntList list);

```
// Returns sublist containing divisors of n
public static intList FilterDivisors(int n, IntList list)
{
     if ( isEmpty(list) )  {          // base case
          return empty();
     } else if (n%head(list) == 0) {  // recursive case
          return prepend(head(list),
                 filterDiviors(n, tail(list)));
     } else {
          return filterDiviors(n, tail(list));
     }
}
```