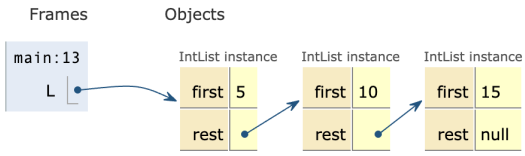
CS61B sp24 week2

tips

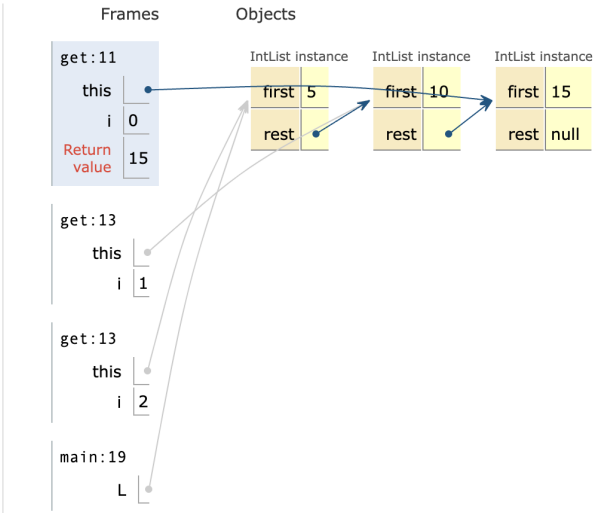
1. a java visualizer web
2. the golden rules of equals: we copy the bits from what set in the right of the equal sign and rewrite them into what in the left of the `=` sign
3. copy the bits = pass by value
   - reference type cannot fit the memory box, we need to use an arrow(reference) to where objects really lives(could be set to null)
4. array and list are not primitive types
   1. `int[] a = new int[]{0, 1, 2, 3, 4}`
      - if the type of elements in array is not primitive (eg. String) it would not store the element, it would store their address
   2. `List<String> lst = new ArrayList<>()`
      - Most implementations of the List interface in Java are mutable like ArrayList and LinkedList
      - but if we use `List.of` to create a list like this `List<Integer> list = List.of(1, 2, 3)` ; The list is immutable.
   3. how to get the length of array/list/set ?
      - **Array**: `int length = array.length;` use the length property
      - **List**: `int size = list.size();`
      - **Set**: `int size = set.size();` use `size()` method
   4. how could we get the specific elements in array/list/set?
      - **Array**: Use the index directly with square brackets (`array[index]`).
      - **List**: Use the get(int index) method (`list.get(index)`).
      - **Set**: Since sets do not support indexed access, use an iterator or convert the set to a list/array first. (`Iterator<Integer> iterator = set.iterator();if (iterator.hasNext()) {int element = iterator.next();`)`
5. the difference between `IntList` and `SLList`
   1. if we using `SLList` , there is no need to specify null which make recursion easier
   2. a typical example for IntList

```java
public class IntList {
    public int first;
    public IntList rest;

    public IntList(int f, IntList r) {
        first = f;
        rest = r;
    }
    public static void main(String[] args) {
       IntList L = new IntList(15, null);
       L = new IntList(10, L);
       L = new IntList(5, L);
    }


}
```



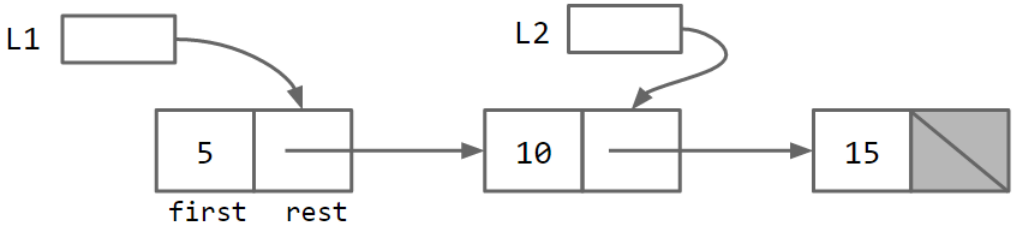
3. the comparison of IntList and SLList, there is the IntList

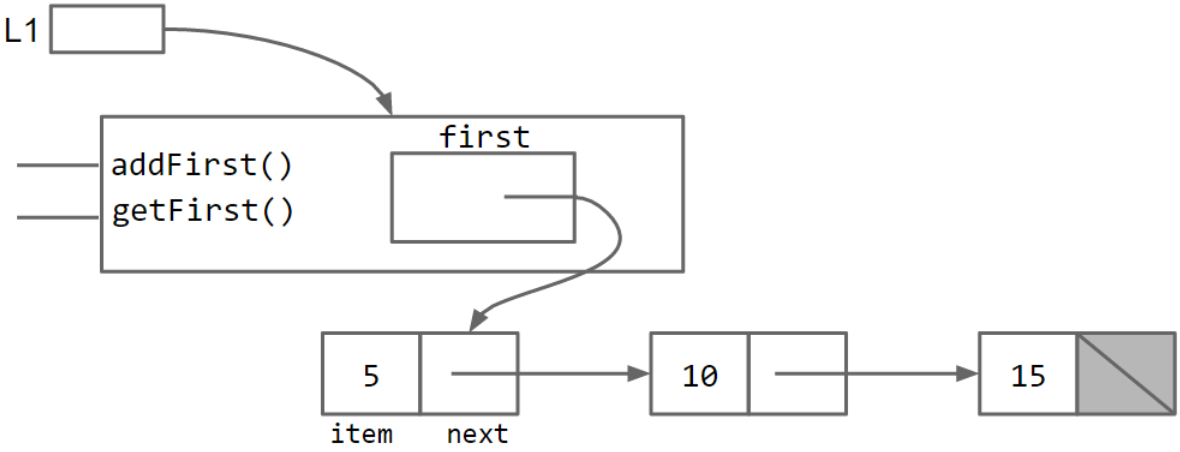
```
3     public IntList rest;
4
5     public IntList(int f, IntList r) {
6         first = f;
7         rest = r;
8     }
9    public int get(int i){
10      if (i == 0){
11        return first;
12      }
13      return rest.get(i - 1);
14   }
15    public static void main(String[] args) {
16       IntList L = new IntList(15, null);
17       L = new IntList(10, L);
18       L = new IntList(5, L);
19       L.get(2);
20    }
21
```



here is [the SLList]

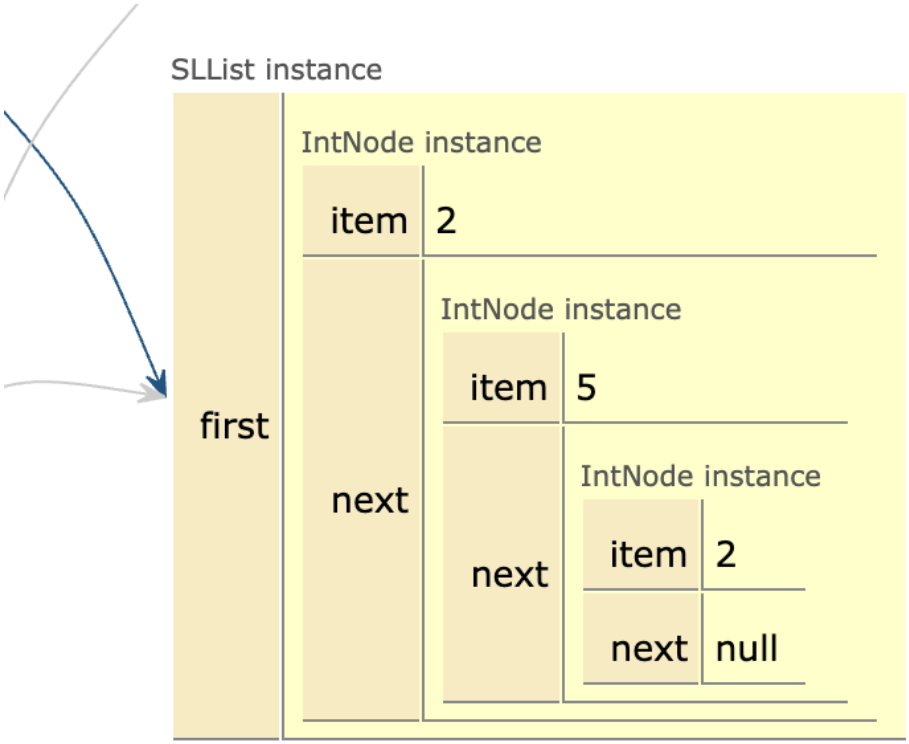
4. so basically the `SLList` class (the `first` instance) acts as a middleman between the list user and the naked recursive data structure.





6. Improvement : Public vs. Private
    1. if we change the public keyword to the private keyword, it would ban the access of the things that out of the current class to get this instance **i.e. Private variables and methods can only be accessed by code inside the same java file**
    2. java has a reflection library that allows you to get the primary instance variable
7. Improvement : Nested Classes
    1. if we set a static class inside another class, the inner class's instance cannot access the instance outside itself anymore
    2. if the instance private but the class is public, we could make this class variables outside, but we could not use it to do anything else

3. NOTICE: item & next could be many in one SLList, but there could only be one SLList eg.



4. Here since the number of parameter of the method are not equal, so the overloaded is allowable

```
/** Returns the size of the list starting at IntNode p. */
private static int size(IntNode p) {
    if (p.next == null) {
        return 1;
    }

    return 1 + size(p.next);
}
```

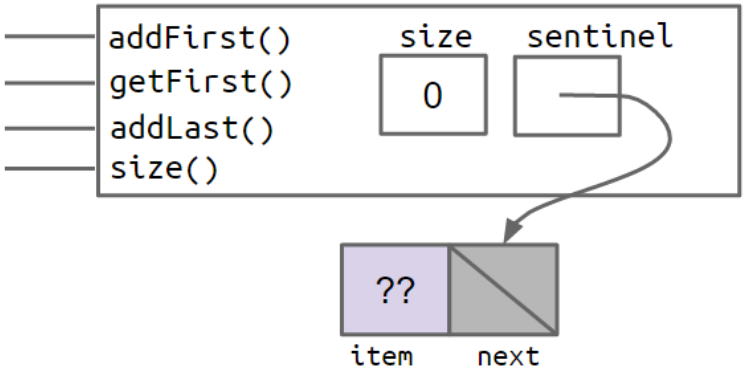
Using this method, we can easily compute the size of the entire list:

```
public int size() {
    return size(first);
}
```

Here, we have two methods, both named `size`. This is allowed in Java, since they have different parameters. We say that two methods with the same name but different signatures are **overloaded**. For more on overloaded methods, see Java's official documentation.

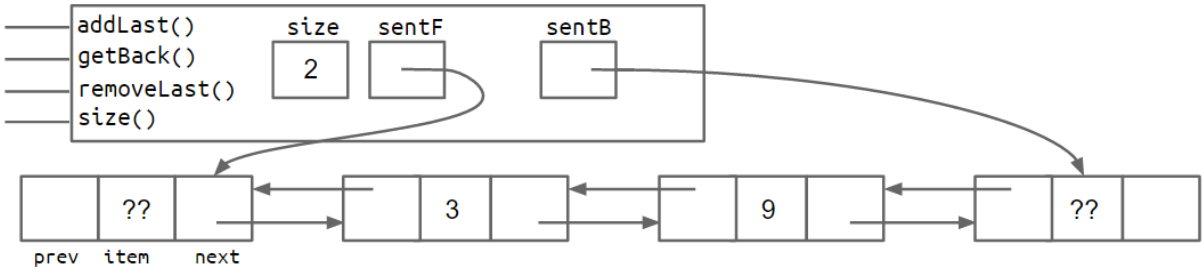
5. keep track of size when user modify the list instead calculating it after the whole SLList has been set, would save much more time
    - and without using SLList, there is no room for size() method in the IntList

8. Improvement : Sentinel node
    - instead of using a specific block to consider the case when the `first == null`, we could set the SLList as the combination of size and sentinel, which promise the first item in SLList would never become null

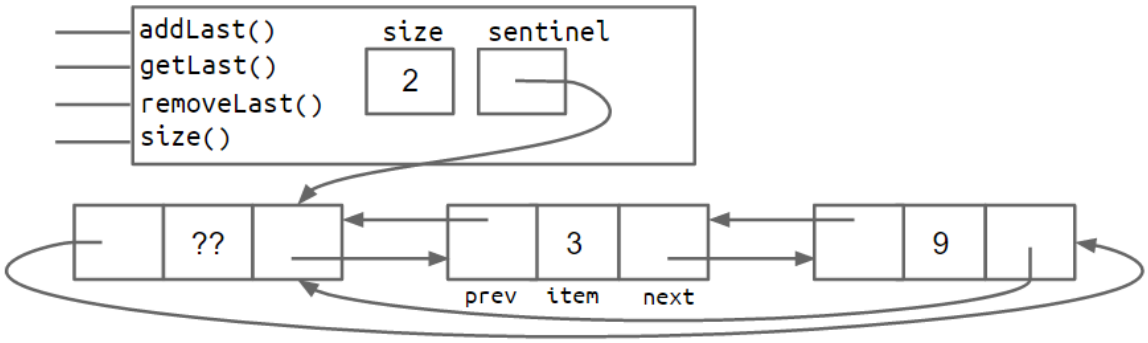


- so the empty list would like this:

9. to solve the slow problem of SLList
   - add a pointer to the end, so that we don't need to iterate the whole list when we need the size, `addlast() || getlast()`
   - but the remove would still be slow, give the second last pinter would still not be enough
   - So we decided to use the **Doubly Linked List** (SL stands for singly-linked) which means we use 2 pointers, one is normal, one start from last to the first elements
   - but how to solve the complexity of this method?
   - one of the solution: double sentinel eg.



   - one of the best solution: build a circular linked list -> the last sentinel.next would back to the first sentinel



   eg.

10. Java allows us to defer type selection until declaration : generics
    - using angle brackets and a placeholder (could be anything like cheese or pineapple)
    - under this circumstance, we use null instead of 0 in integers
    - then when we need to use need we just need to write the variable type replace the placeholder like this `DLList<Integer> d1 = new DLList<Integer>(5);` or `DLList<String> d2 = new DLList<>("hello");`
    - if we don't specify we cannot put multiple variables on it!!!
    - do not put the primitive type like `Integer` on it!
    - The right side of the equal sign can infer the type from the empty angle brackets, matching the type of the variable on the left side.
11. Array
    - DEF: a special kind of object which consists of a **numbered** sequence of memory boxes, which means all the memory boxes do not have the names, instead they have corresponding numbers
    - it consists of a **fixed** length (integer) and a sequence of N memory boxes (N = length) *length is a public property/attribute here*

- the diff between arrays and classes: (they both have fixed nummber of boxes)
- array must contain same variable types
- array indices can be computed at runtime eg. `int k = x[indexOfInterest];` this is dynamic indices
- **NOTICE**: string is a reference type!
- `arraycopy` is a faster syntax to copy the array here is the syntax
- `System.arraycopy(b, 0, x, 3, 2);`
- = (In Python): `x[3:5] = b[0:2]`
- 3 valid notations:

```
x = new int[3];
y = new int[]{1, 2, 3, 4, 5};
int[] z = {9, 10, 11, 12, 13};
```