Quiz 2 (Time- 5 minutes)

```
Object ReverseLL(Object Head, int n)
{
   ------
}
```

Time Complexity T(n)=? Justify

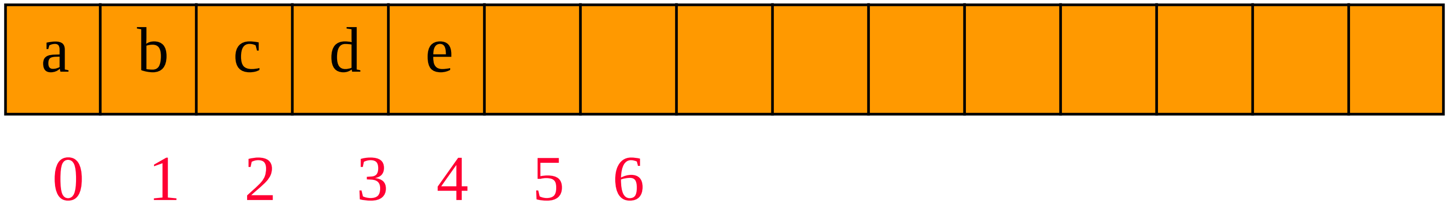
Stacks

```
public interface Stack
{
    public boolean empty();
    public Object peek();
    public void push(Object theObject);
    public Object pop();
}
```

Derive from a linear list class

- ArrayLinearList
- LinkedList

Derive from ArrayLinearList

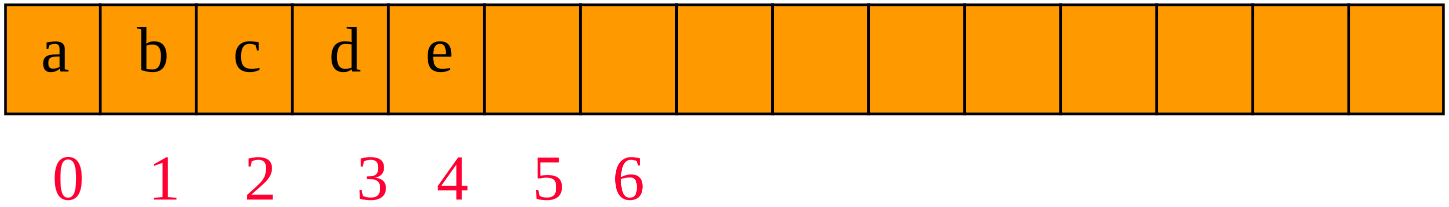


a b c d e

0 1 2 3 4 5 6

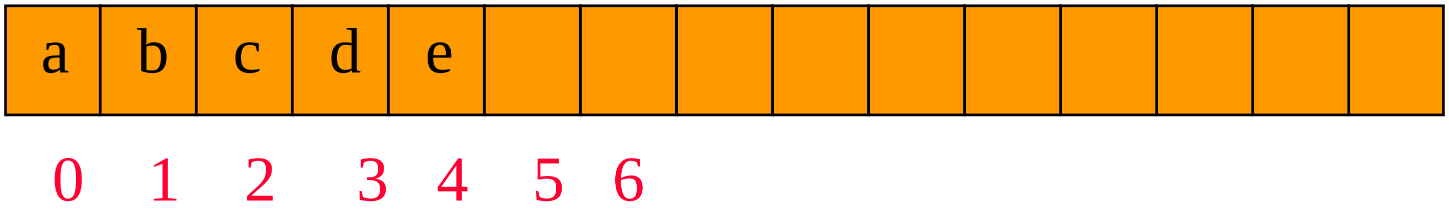
- stack top is either left end or right end of linear list
- empty() => isEmpty()
  - O(1) time
- peek() => get(0) or get(size() - 1)
  - O(1) time

Derive from ArrayLinearList



- when top is left end of linear list
  - push(theObject) => add(0, theObject)
    - O(size) time
  - pop() => remove(0)
    - O(size) time

Derive from ArrayLinearList

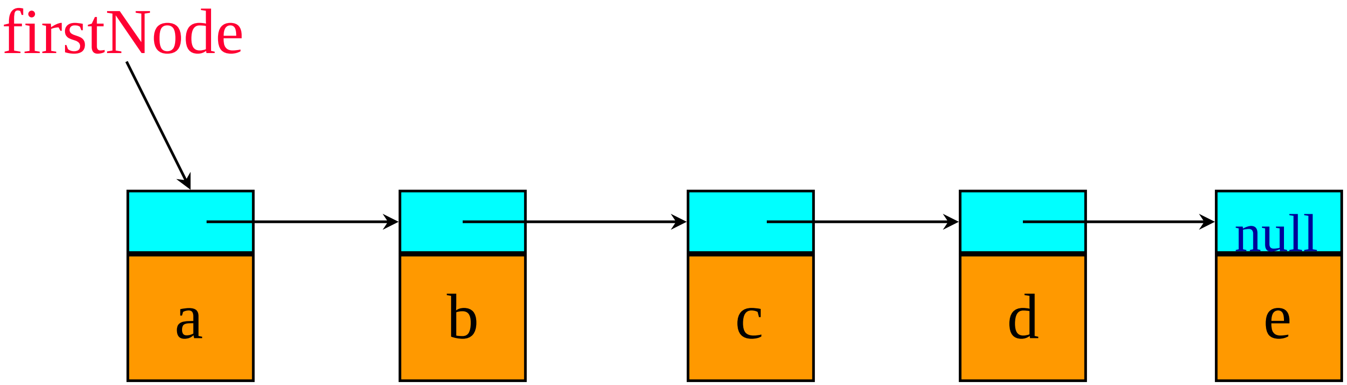


a b c d e

0 1 2 3 4 5 6

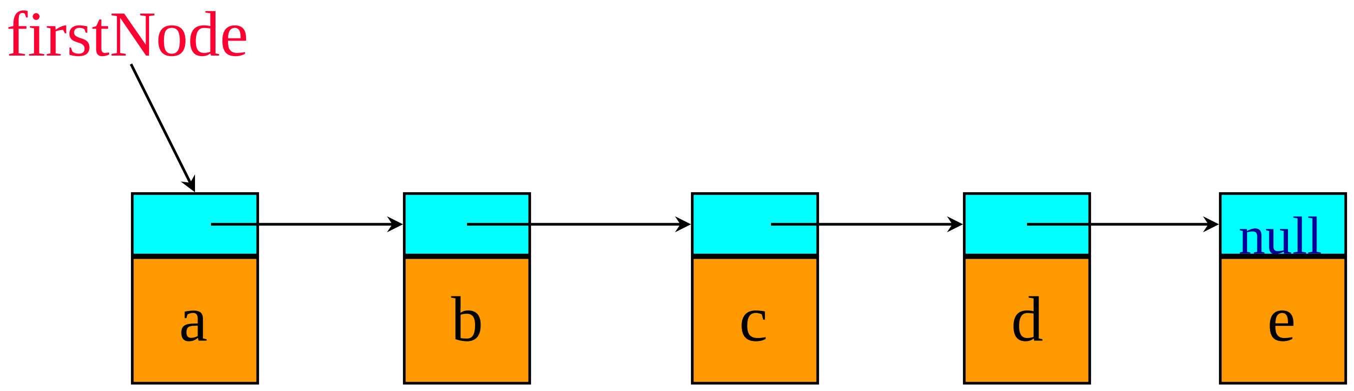
- when top is right end of linear list
  - push(theObject) => add(size(), theObject)
  - O(1) time
  - pop() => remove(size()-1)
  - O(1) time
- use right end of list as top of stack

Derive from LinkedList



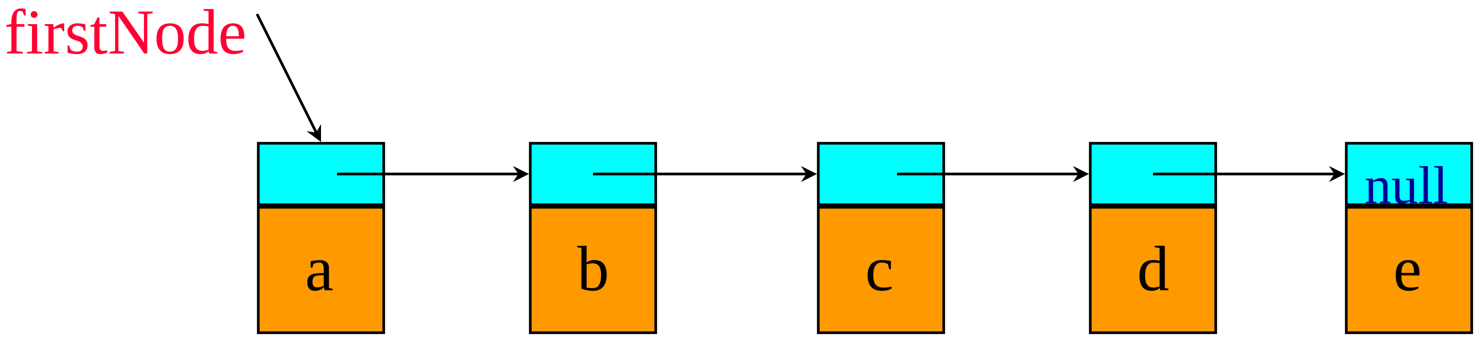
- stack top is either left end or right end of linear list
- empty() => isEmpty()
  - O(1) time

Derive from LinkedList



- when top is left end of linear list
  - peek() => get(0)
    - O(1) time
  - push(theObject) => add(0, theObject)
    - O(1) time
  - pop() => remove(0)
    - O(1) time

Derive from LinkedList



- when top is right end of linear list
  - peek() => get(size() - 1)
    - O(size) time
  - push(theObject) => add(size(), theObject)
    - O(size) time
  - pop() => remove(size()-1)
    - O(size) time
- use left end of list as top of stack

Derive from ArrayLinearList

```
import java.util.*;  // has stack exception

public class DerivedArrayStack
                 extends ArrayLinearList
                 implements Stack
{
  // constructors come here
  // Stack interface methods come here
}
```

Constructors

```
/** create a stack with the given initial capacity */
public DerivedArrayStack(int initialCapacity)
  {super(initialCapacity);}

/** create a stack with initial capacity 10 */
public DerivedArrayStack()
  {this(10);}
```

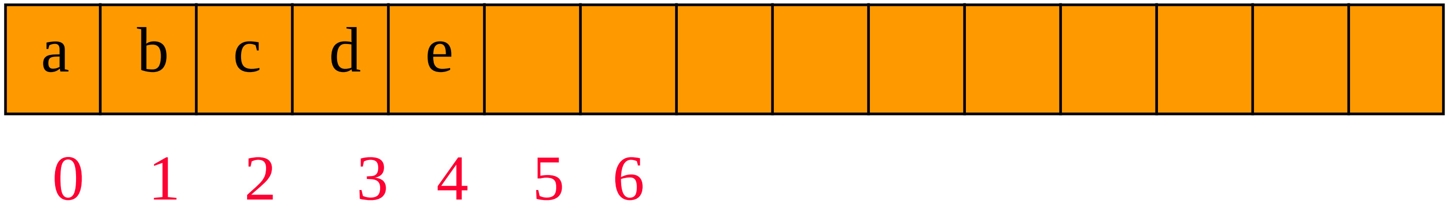
empty()   and   peek()

a	b	c	d	e										
0	1	2	3	4	5	6								

```
public boolean empty()
  {return isEmpty();}

public Object peek()
{
  if (empty())
     throw new EmptyStackException();
  return get(size() - 1);
}
```

push(theObject) and pop()



```
public void push(Object theElement)
  {add(size(), theElement);}


public Object pop()
{
  if (empty())
     throw new EmptyStackException();
  return remove(size() - 1);
}
```

Evaluation

- Merits of deriving from ArrayLinearList
  - Code for derived class is quite simple and easy to develop.
  - Code is expected to require little debugging.
  - Code for other stack implementations such as a linked implementation are easily obtained.
    - Just replace extends ArrayLinearList with extends LinkedList
    - For efficiency reasons we must also make changes to use the left end of the list as the stack top rather than the right end.

Demerits

- All public methods of ArrayLinearList may be performed on a stack.
  - get(0) … get bottom element
  - remove(5)
  - add(3, x)
  - So we do not have a true stack implementation.
  - Must override undesired methods.

```
public Object get(int theIndex)
  {throw new UnsupportedOperationException();}
```

Change earlier use of get(i) to super.get(i).

Demerits

- Unecessary work is done by the code.
  - peek() verifies that the stack is not empty before get is invoked. The index check done by get is, therefore, not needed.
  - add(size(), theElement) does an index check and a for loop that is not entered. Neither is needed.
  - pop() verifies that the stack is not empty before remove is invoked. remove does an index check and a for loop that is not entered. Neither is needed.
  - So the derived code runs slower than necessary.

Evaluation

- Code developed from scratch will run faster but will take more time (cost) to develop.
- Tradeoff between software development cost and performance.
- Tradeoff between time to market and performance.
- Could develop easy code first and later refine it to improve performance.

A faster pop()

```
if (empty())
    throw new EmptyStackException();
  return remove(size() - 1);
```

vs.

```
try {return remove(size() - 1);}
catch(IndexOutOfBoundsException e)
   {throw new EmptyStackException();}
```

Code from scratch

- Use a 1D array stack whose data type is Object.
  - same as using array element in ArrayLinearList
- Use an int variable top.
  - Stack elements are in stack[0:top].
  - Top element is in stack[top].
  - Bottom element is in stack[0].
  - Stack is empty iff top = -1.
  - Number of elements in stack is top+1.

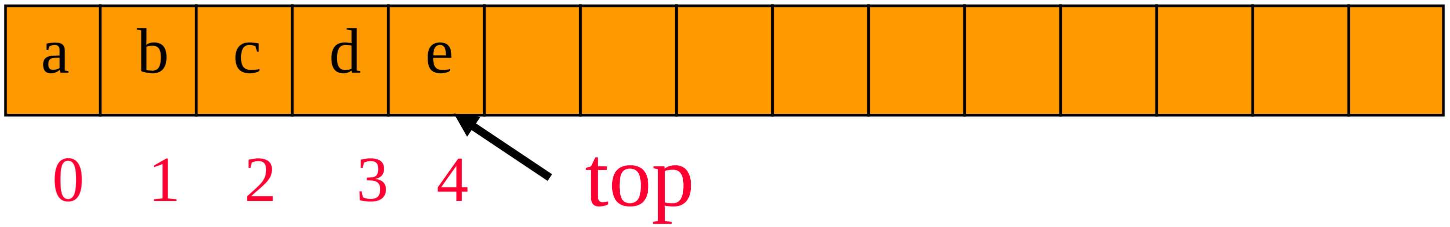
Code from scratch

```
import java.util.EmptyStackException;
import utilities.*;  // ChangeArrayLength
public class ArrayStack implements Stack
{
   // data members
   int top;                // current top of stack
   Object [] stack;  // element array
   // constructors come here
   // Stack interface methods come here
}
```

Constructors

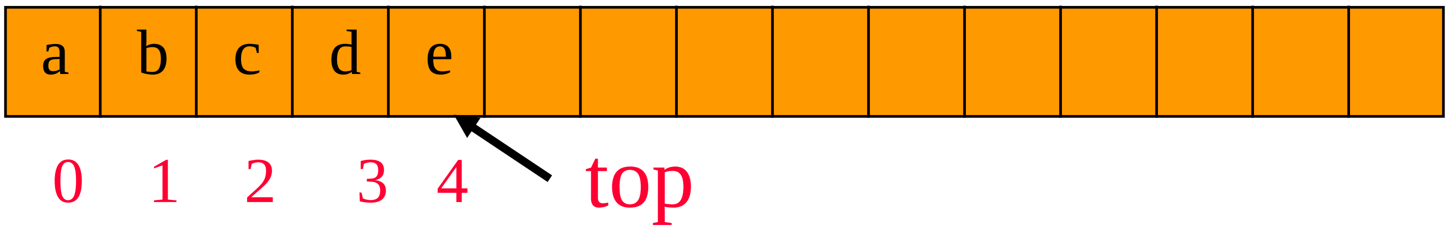
```
public ArrayStack(int initialCapacity)
{
  if (initialCapacity < 1)
     throw new IllegalArgumentException
      ("initialCapacity must be >= 1");
  stack = new Object [initialCapacity];
  top = -1;
}
public ArrayStack()
  {this(10);}
```

push(…)



```
public void push(Object theElement)
{
  // increase array size if necessary
   if  (top == stack.length - 1)
       stack = ChangeArrayLength.changeLength1D
                 (stack, 2  * stack.length);
  // put theElement at the top of the stack
  stack[++top] = theElement;
}
```

pop()



a b c d e

0 1 2 3 4

top

```
public Object pop()
{
 if (empty())
    throw new EmptyStackException();
 Object topElement = stack[top];
 stack[top--] = null;   // enable garbage collection
 return topElement;
}
```

java.util.Stack

- Derives from java.util.Vector.
- java.util.Vector is an array implementation of a linear list.