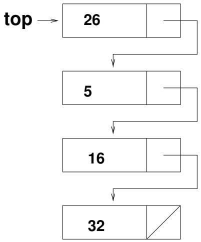
Linked List Implementation of a Stack



The `Stack` interface can be implemented by using a linked list in which the top of the stack is represented by the first object in the list.

Stack Operations

To implement the push function, we create a new node in the list and attach it as the new first element. To implement the pop function, we advance the top of the stack to the next object in the list (if there is one). An empty stack is represented by an empty linked list.

To implement the class Stack we use the adapter design pattern. It maps the Stack functions onto the LinkedList class functions.

Stack functions	LinkedList functions
`isEmpty()`	`isEmpty()`
`top()`	`first()`
`push(e)`	`insertFirst(e)`
`pop()`	`removeFirst()`

The LinkedStack Class

```
class LinkedStack {
private: LinkedList ll;
public:
   class StackEmptyException : public RuntimeException {
      public: StackEmptyException(const string& err) :
                                 RuntimeException(err) {}
   };
   LinkedStack() : ll() { } // default constructor
   bool isEmpty() const { return ll.isEmpty(); }
   void push(const int elem) { ll.insertFirst(elem); }
   int pop() throw(StackEmptyException);
   int top() const throw(StackEmptyException);
};
```

LinkedStack Implementation

```
int LinkedStack::pop() throw(StackEmptyException) {
   if (ll.isEmpty())
      throw StackEmptyException("Stack is empty");
   return ll.removeFirst();
}
int LinkedStack::top() const throw(StackEmptyException) {
   if (ll.isEmpty())
      throw StackEmptyException("Stack is empty");
   return ll.first();
}
```

LinkedStack Implementation (cont.)

```
int LinkedStack::pop() throw(StackEmptyException) {
   if (ll.isEmpty())
      throw StackEmptyException("Stack is empty");
   return ll.removeFirst();
}
int LinkedStack::top() const throw(StackEmptyException) {
   if (ll.isEmpty())
      throw StackEmptyException("Stack is empty");
   return ll.first();
}
```