Linked List (Simple Version)

Đoạn code java phía dưới là thiết kế chi tiết của lớp LinkedList (Dạng đơn giản), và phương thức main để kiểm tra. Các em hãy hoàn thiện lớp này với gợi ý sau:

LinkedList<T>: Là kiểu dữ liệu Generic, cho phép lưu trữ tập hợp các phần tử kiểu T. LinkedList<T>, như tên đã thể hiện, lưu trữ dữ liệu ở dạng danh sách liên kết đơn (Tài liệu tham khảo, Chapter 3), hỗ trợ các methods như trong file thiết kế (Java và C#).

Input

+ Dòng đầu tiên gồm số 0, và m (0 <= m <= 10000) là số lượng câu lệnh (command)

+ m dòng tiếp theo chứa các câu lệnh có dạng: "insertAt 0 value" (Chèn value vào đầu danh sách), hoặc "getAt index" (Lấy ra phần tử tại vị trí index), index và value là các số nguyên 32 bit.

Output

+ Với các command getAt, xuất kết quả vào standard output. Nếu chỉ số nhập không hợp lệ, xuất ra chuỗi "null" (no quoute)

Example

Input:
0 10
insertAt 0 82
insertAt 0 420
insertAt 0 -156
insertAt 0 -796
getAt 0
insertAt 0 -97
insertAt 0 -514
insertAt 0 -541
getAt 5
insertAt 0 -700

Output:
-796
420

Program

```
import java.util.Scanner;

class LinkedList<T extends Number> {

  static private class LinkedNode<U extends Number> {
    U number;
    LinkedNode<U> next;

    public LinkedNode(U number) {
      this.number = number;
```

```
    }
  }

  LinkedNode<T> head = null;

  public void insertAt(int index, T number) {
    // Your code here
  }

  /**
   * @return null if index is out of range
   */
  public T getAt(int index) {
    // Your code here
    return null;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    LinkedList<Integer> linkedList = new LinkedList<Integer>();
    // You code here
  }
}
```

```
using System;

namespace DSA17W
{
    class EILINKEA
    {
        static void Main(string[] args)
        {
            NextInt();
            var m = NextInt();

            LinkedList<int> linkedList = new LinkedList<int>();
            for (int i = 0; i < m; i++)
            {
                var command = Next();
                if (command == "insertAt")
                {
                }
                else if (command == "getAt")
                {
                }
            }
        }

        #region Input Wrapper
        static int s_index = 0;
        static string[] s_tokens;

        private static string Next()
        {
            while (s_tokens == null || s_index == s_tokens.Length)
            {
                s_tokens = Console.ReadLine().Split(new char[] { ' ', '\t', '\n' }, StringSplitOptions.RemoveEmptyEntries);
                s_index = 0;
            }
```

```
            return s_tokens[s_index++];
        }

        private static int NextInt()
        {
            return Int32.Parse(Next());
        }

        #endregion
    }

    class LinkedList<T>
    {
        class LinkedNode<U>
        {
            public U Item { get; set; }
            public LinkedNode<U> Next { get; set; }

            public LinkedNode(U item)
            {
                Item = item;
            }
        }

        LinkedNode<T> head = null;
        public int Count { get; set; }

        public void InsertAt(int index, T item)
        {
            // Your code here
        }

        public T GetAt(int index)
        {
            if (index < 0 || index >= Count)
            {
                throw new IndexOutOfRangeException();
            }
            // Your code here
            return head.Item;
        }
    }
}
```