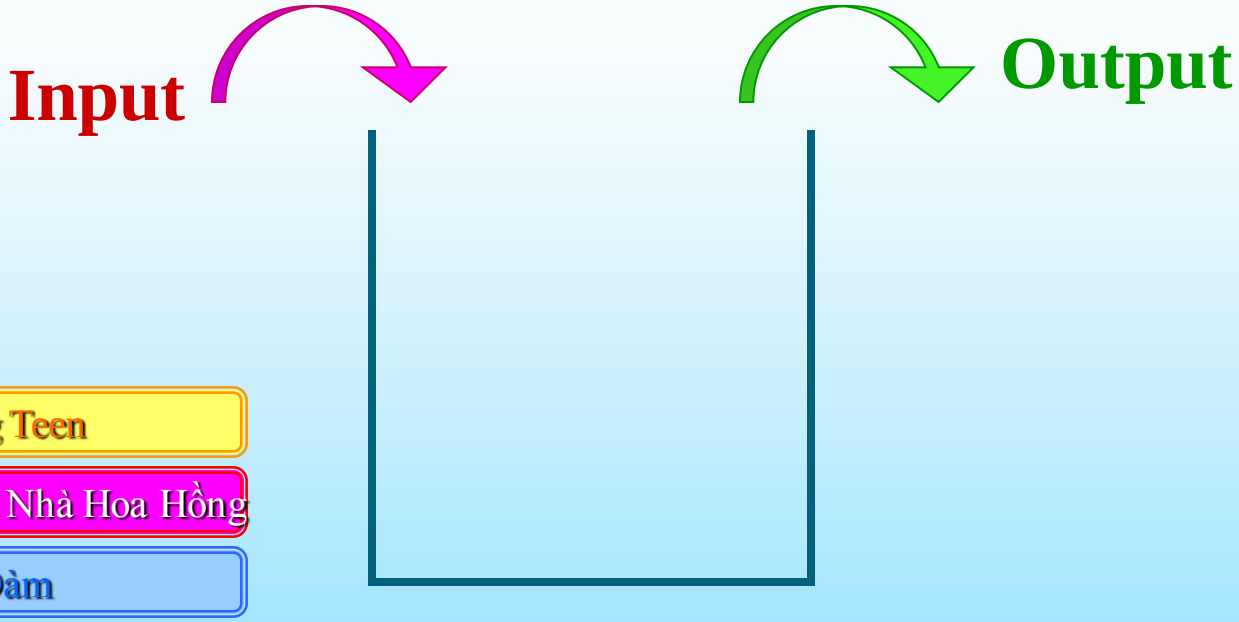
STACK

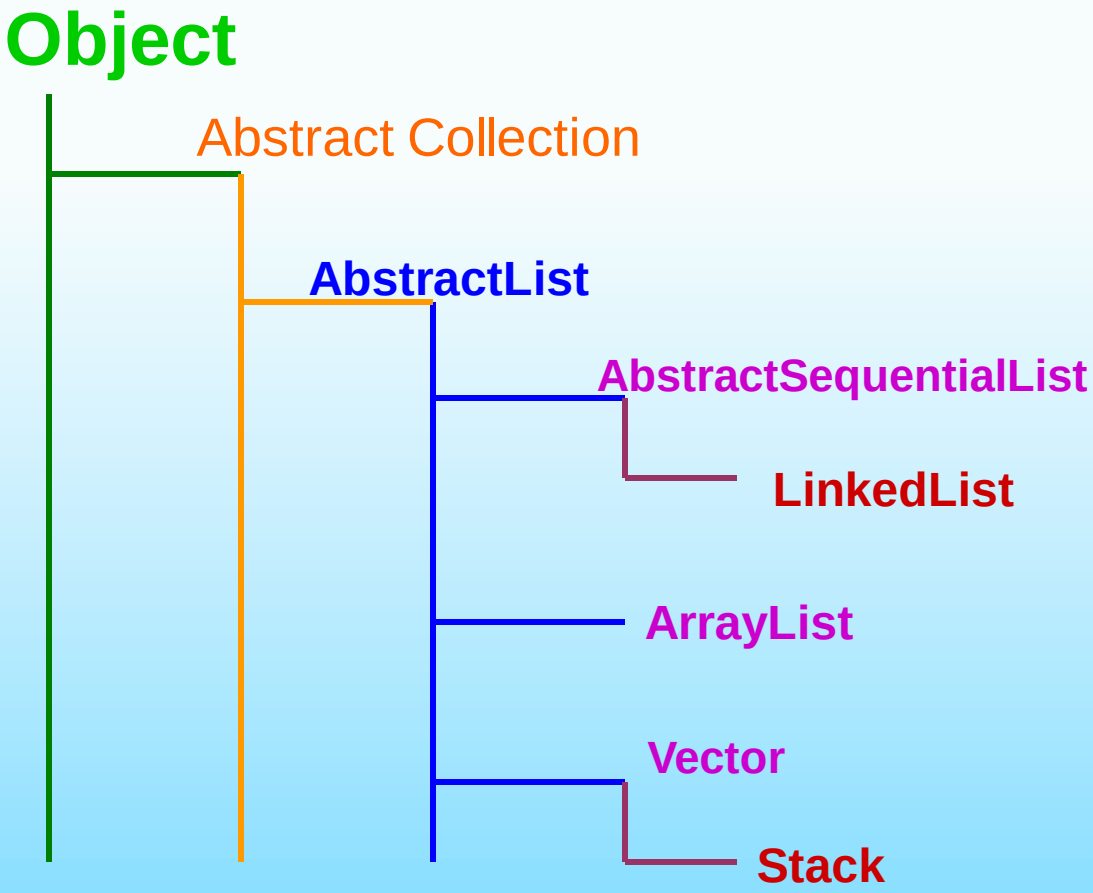
Ví dụ



Định nghĩa Stack

- Stack là vật chứa mà phần tử được thêm vào và bỏ ra tại vị trí đầu tiên của vật chứa.
- Stack tuân theo luật *last-in*, *first-out* or *LIFO* .

Stack trong Java Collection framework



KHỞI TẠO VÀ PHƯƠNG THỨC

- Khởi tạo Stack

  import java.util.Stack;

  ….

  Stack s = new Stack();

- Thêm 1 phần tử vào Stack

  s.push(“learn Stack”);

  s.push(“practise Stack”);

- Loại bỏ 1 phần tử ra khỏi Stack

  s.pop();

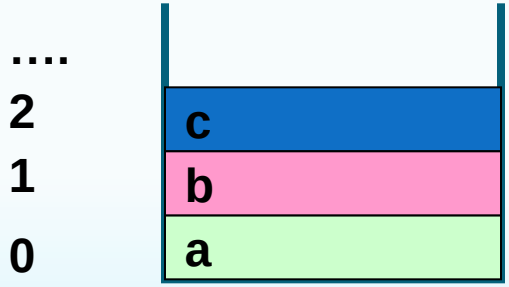
KHỞI TẠO VÀ PHƯƠNG THỨC

- Ý nghĩa của các phương thức:
  - ✓ empty() return true if no element in stack.
  - ✓ **peek()** , firstElement() return top element of stack but not delete top element in stack.
  - ✓ **pop()** return top element of stack and delete top element in stack
  - ✓ **push(Object obj)**, add(Object obj) push the element in stack
  - ✓ elements() return iterator of elements
  - ✓ size() return the size of stack

Our Stack ADT	Class java.util.Stack	
size( )	size( )	
isEmpty( )	empty( )	⇐
push(e)	push(e)	
pop( )	pop( )	
top( )	peek( )	⇐

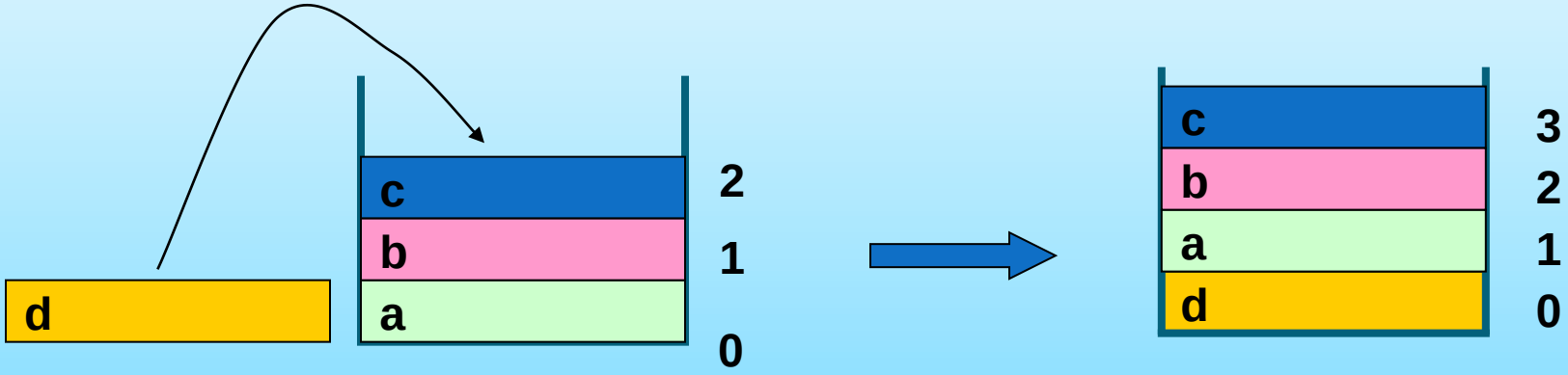
KHỞI TẠO VÀ PHƯƠNG THỨC

- ✓ search(Object obj) return index of element in stack



s.search("a") → 0

- ✓ add (int position, Object obj)



s.add(0,"d")

KHỞI TẠO VÀ PHƯƠNG THỨC

- remove(int position) remove an object at input position.
- remove(Object obj) remove an object in Stack that equal input object.

  This method is only true if each value of object in stack is unique.

- Challenge: How do you fix the error of this method? (rewrite this method)
- removeAllElements() return a empty Stack

XÂY DỰNG STACK TỪ ARRAY

```
1  public class ArrayStack<E> implements Stack<E> {
2    public static final int CAPACITY=1000;  // default array capacity
3    private E[ ] data;                      // generic array used for storage
4    private int t = −1;                     // index of the top element in stack
5    public ArrayStack( ) { this(CAPACITY); } // constructs stack with default capacity
6    public ArrayStack(int capacity) {       // constructs stack with given capacity
7      data = (E[ ]) new Object[capacity];   // safe cast; compiler may give warning
8    }
9    public int size( ) { return (t + 1); }
10   public boolean isEmpty( ) { return (t == −1); }
11   public void push(E e) throws IllegalStateException {
12     if (size( ) == data.length) throw new IllegalStateException("Stack is full");
13     data[++t] = e;                        // increment t before storing new item
14   }
15   public E top( ) {
16     if (isEmpty( )) return null;
17     return data[t];
18   }
19   public E pop( ) {
20     if (isEmpty( )) return null;
21     E answer = data[t];
22     data[t] = null;                       // dereference to help garbage collection
23     t−−;
24     return answer;
25   }
26 }
```

Code Fragment 6.2: Array-based implementation of the Stack interface.

XÂY DỰNG STACK TỪ SINGLY LINKED LIST

Stack Method	*Singly Linked List Method*
size( )	list.size( )
isEmpty( )	list.isEmpty( )
push(*e*)	list.addFirst(*e*)
pop( )	list.removeFirst( )
top( )	list.first( )

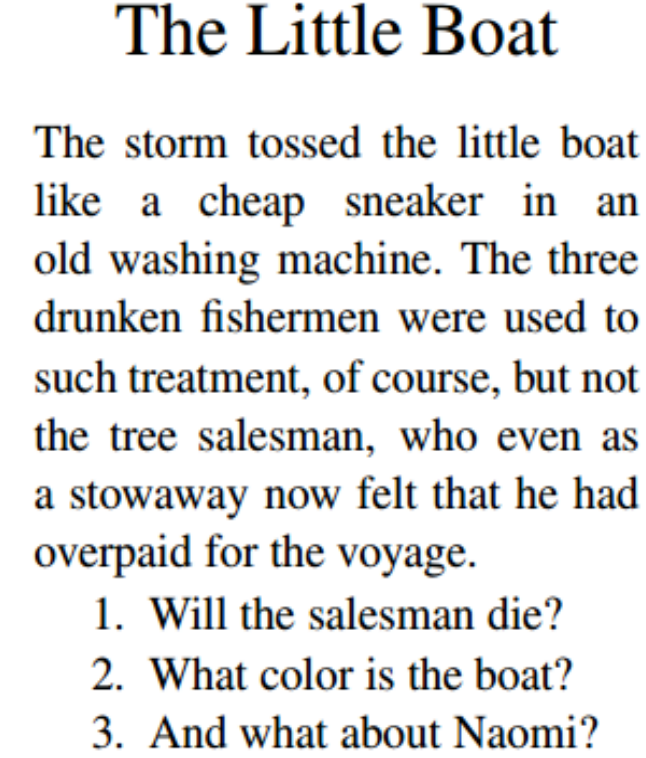
```
1  public class LinkedStack<E> implements Stack<E> {
2    private SinglyLinkedList<E> list = new SinglyLinkedList<>();    // an empty list
3    public LinkedStack( ) { }                  // new stack relies on the initially empty list
4    public int size( ) { return list.size( ); }
5    public boolean isEmpty( ) { return list.isEmpty( ); }
6    public void push(E element) { list.addFirst(element); }
7    public E top( ) { return list.first( ); }
8    public E pop( ) { return list.removeFirst( ); }
9  }
```

Code Fragment 6.4: Implementation of a Stack using a SinglyLinkedList as storage.

BÀI TẬP HTML

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine.  The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

(a)



(b)

Figure 6.3: Illustrating (a) an HTML document and (b) its rendering.

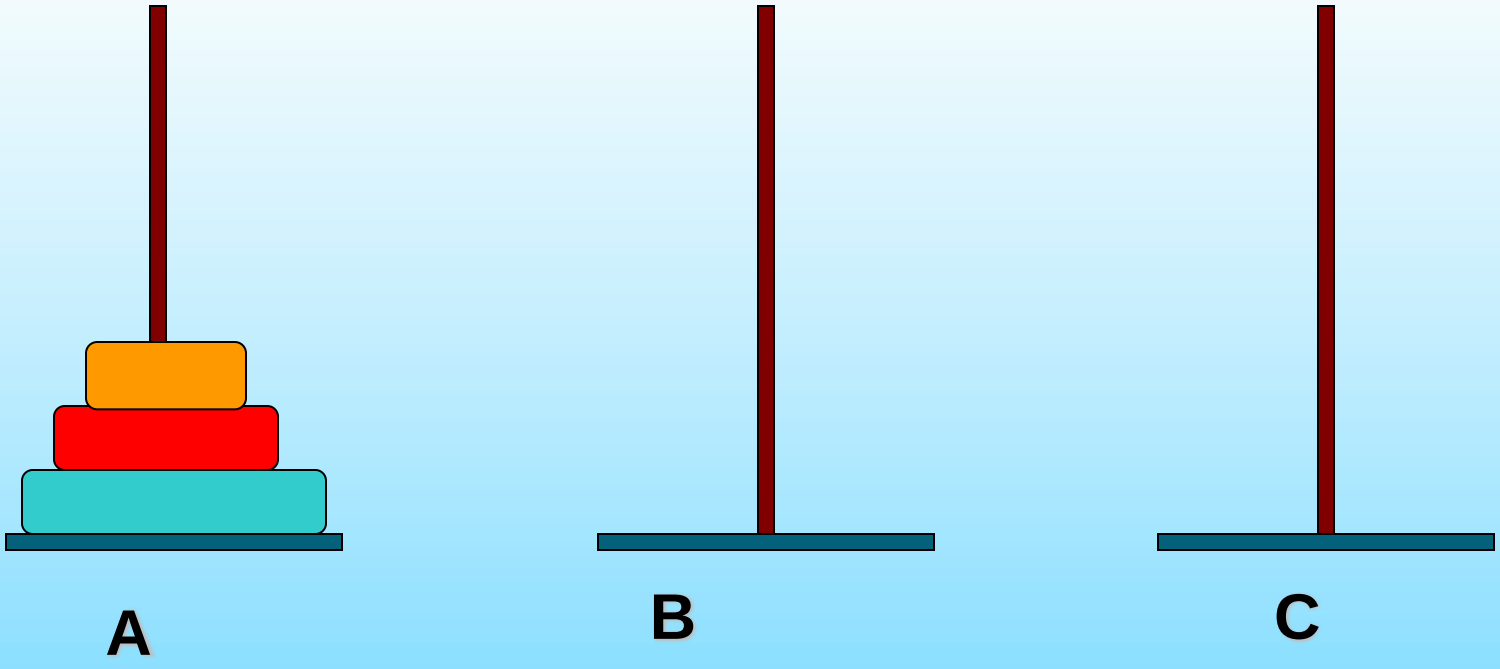
BÀI TẬP HTML

```
/** Tests if every opening tag has a matching closing tag in HTML string. */
public static boolean isHTMLMatched(String html) {
  Stack<String> buffer = new LinkedStack<>();
  int j = html.indexOf('<');                    // find first '<' character (if any)
  while (j != -1) {
    int k = html.indexOf('>', j+1);             // find next '>' character
    if (k == -1)
      return false;                             // invalid tag
    String tag = html.substring(j+1, k);        // strip away < >
    if (!tag.startsWith("/"))                   // this is an opening tag
      buffer.push(tag);
    else {                                      // this is a closing tag
      if (buffer.isEmpty())
        return false;                           // no tag to match
      if (!tag.substring(1).equals(buffer.pop()))
        return false;                           // mismatched tag
    }
    j = html.indexOf('<', k+1);                 // find next '<' character (if any)
  }
  return buffer.isEmpty();                      // were all opening tags matched?
}
```

Code Fragment 6.8: Method for testing if an HTML document has matching tags.

BÀI TẬP THÁP HÀ NỘI

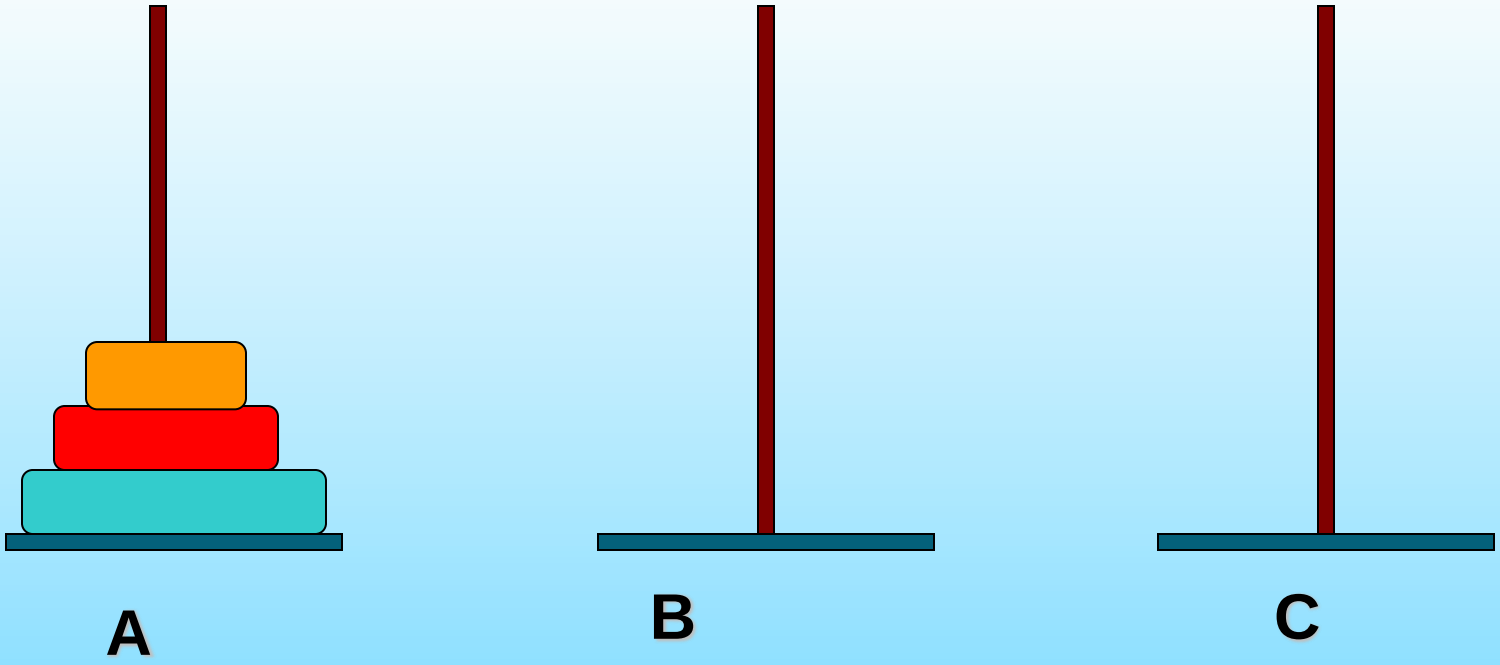
- Trò chơi tháp Hà Nội



Làm thế nào để di chuyển 3 dĩa từ cột A sang cột C với số bước di chuyển ít nhất?

Ứng dụng

- Giải pháp



BÀI TẬP THÁP HÀ NỘI

```
FUNCTION MoveTower(disk, source, dest, spare):
IF disk == 0, THEN:
    move disk from source to dest
ELSE:
    MoveTower(disk - 1, source, spare, dest)   // Step 1 above
    move disk from source to dest              // Step 2 above
    MoveTower(disk - 1, spare, dest, source)   // Step 3 above
END IF
```

Cách giải

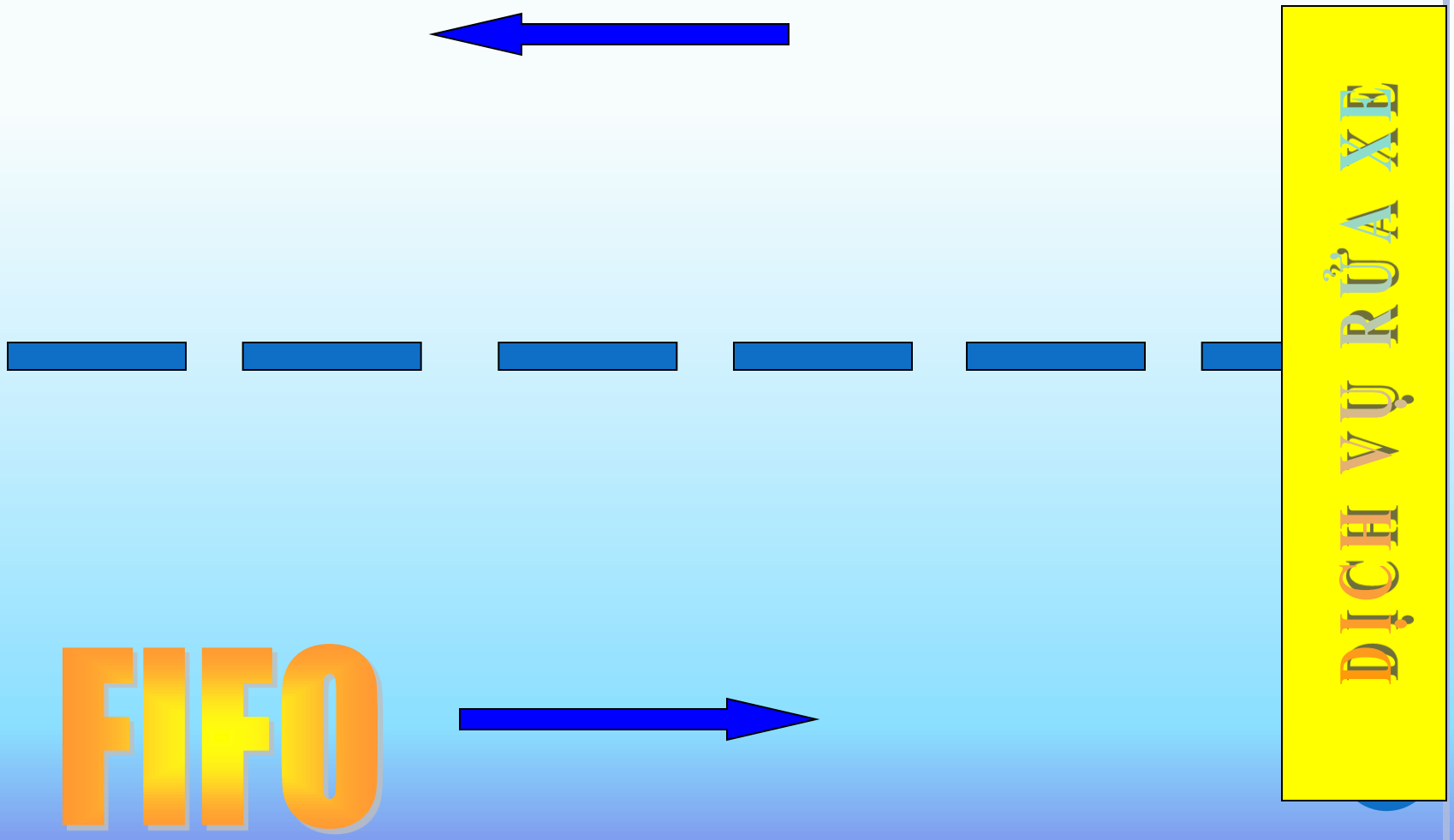
- Xây dựng mỗi cọc là một Stack chứa các đĩa có các trong số khác nhau, đĩa có trọng số nhỏ hơn sẽ ở trên, đĩa có trong số lớn hơn sẽ ở dưới.
- Sử dụng giải thuật trên để di chuyển đĩa

Queue

Queue

- Queue là vật chứa mà phần tử mới được thêm vào và loại bỏ ra tại vị trí cuối.
- Queue tuân theo quy luật *first-come, first-served* (FIFO)

Ví dụ Queue



PHƯƠNG THỨC QUEUE

- a queue is a *first-in, first-out* or *FIFO* data structure.
- comprises all the methods inherited from Container plus the three methods: getHead(), enqueue(), and dequeue()

Ứng dụng Queue

- Xếp hàng chờ phục vụ đồ ăn tại 1 cửa hàng. Người đến trước sẽ được phục vụ trước.

XÂY DỰNG QUEUE TỪ MẢNG

```
/** Implementation of the queue ADT using a fixed-length array. */
public class ArrayQueue<E> implements Queue<E> {
  // instance variables
  private E[ ] data;                          // generic array used for storage
  private int f = 0;                          // index of the front element
  private int sz = 0;                         // current number of elements

  // constructors
  public ArrayQueue( ) {this(CAPACITY);} // constructs queue with default capacity
  public ArrayQueue(int capacity) {           // constructs queue with given capacity
    data = (E[ ]) new Object[capacity];       // safe cast; compiler may give warning
  }

  // methods
  /** Returns the number of elements in the queue. */
  public int size( ) { return sz; }

  /** Tests whether the queue is empty. */
  public boolean isEmpty( ) { return (sz == 0); }

  /** Inserts an element at the rear of the queue. */
  public void enqueue(E e) throws IllegalStateException {
    if (sz == data.length) throw new IllegalStateException("Queue is full");
    int avail = (f + sz) % data.length;       // use modular arithmetic
    data[avail] = e;
    sz++;
  }

  /** Returns, but does not remove, the first element of the queue (null if empty). */
  public E first( ) {
    if (isEmpty( )) return null;
    return data[f];
  }

  /** Removes and returns the first element of the queue (null if empty). */
  public E dequeue( ) {
    if (isEmpty( )) return null;
    E answer = data[f];
    data[f] = null;                           // dereference to help garbage collection
    f = (f + 1) % data.length;
    sz--;
    return answer;
  }
```

Code Fragment 6.10: Array-based implementation of a queue.

XÂY DỰNG QUEUE TỪ SINGLY LINKED LIST

```
/** Realization of a FIFO queue as an adaptation of a SinglyLinkedList. */
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();    // an empty list
  public LinkedQueue( ) { }                    // new queue relies on the initially empty list
  public int size( ) { return list.size( ); }
  public boolean isEmpty( ) { return list.isEmpty( ); }
  public void enqueue(E element) { list.addLast(element); }
  public E first( ) { return list.first( ); }
  public E dequeue( ) { return list.removeFirst( ); }
}
```

Circular Queue

Bài đọc thêm

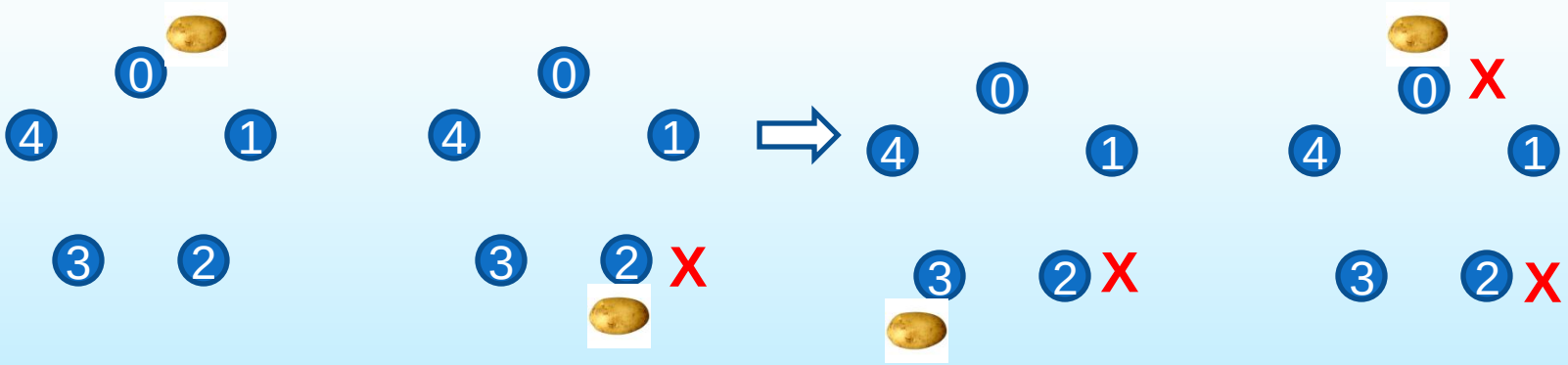
CIRCULAR QUEUE

```
public interface CircularQueue<E> extends Queue<E> {
  /**
   * Rotates the front element of the queue to the back of the queue.
   * This does nothing if the queue is empty.
   */
  void rotate();
}
```

Josephus Problem - Example

M=2, N=5

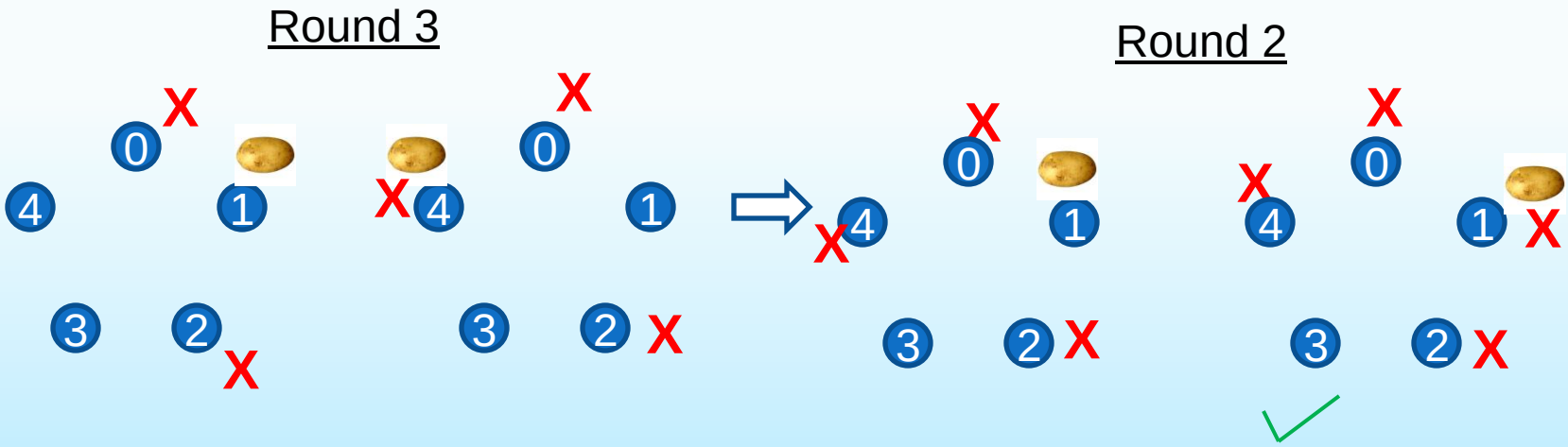
Initial state: Round 1 Round 2



Person removed so far: {2, 0,

Josephus Problem - Example

M=2, N=5



Person removed so far: {2, 0, 4, 1 }

Winner is 3

THE JOSEPHUS PROBLEM

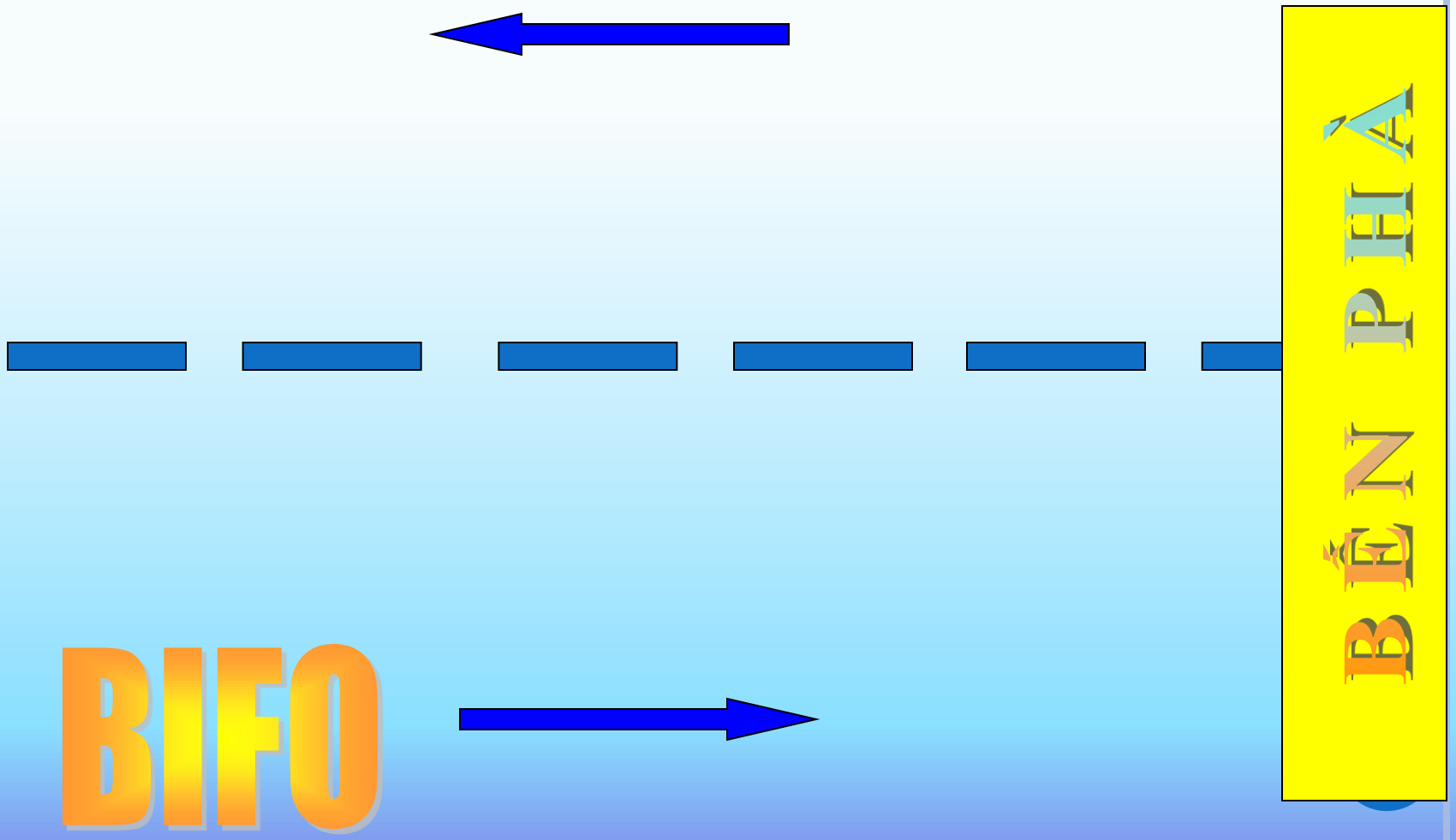
```
public class Josephus {
  /** Computes the winner of the Josephus problem using a circular queue. */
  public static <E> E Josephus(CircularQueue<E> queue, int k) {
    if (queue.isEmpty( )) return null;
    while (queue.size( ) > 1) {
      for (int i=0; i < k−1; i++)    // skip past k-1 elements
        queue.rotate( );
      E e = queue.dequeue( );        // remove the front element from the collection
      System.out.println("       " + e + " is out");
      }
    return queue.dequeue( );         // the winner
  }

  /** Builds a circular queue from an array of objects. */
  public static <E> CircularQueue<E> buildQueue(E a[ ]) {
    CircularQueue<E> queue = new LinkedCircularQueue<>( );
    for (int i=0; i<a.length; i++)
      queue.enqueue(a[i]);
    return queue;
  }
```

Priority Queue

Bài đọc thêm

Priority Queue



ĐỊNH NGHĨA

- priority queue retrieves elements in sorted order after they were inserted in arbitrary order. That is, whenever you call the remove method, you get the smallest element currently in the priority queue. However, the priority queue does not sort all its elements. If you iterate over the elements, they are not necessarily sorted. The priority queue makes use of an elegant and efficient data structure, called a heap. A heap is a self-organizing binary tree in which the add and remove operations cause the smallest element to gravitate to the root, without wasting time on sorting all elements.
- Just like a treeSet, a priority queue can either hold elements of a class that implements the Comparable interface or a Comparator object you supply in the constructor.
- A typical use for a priority queue is job scheduling. Each job has a priority. Jobs are added in random order. Whenever a new job can be started, the highest-priority job is removed from the queue. (Since it is traditional for priority 1 to be the "highest" priority, the remove operation yields the minimum element.)

CÁCH THỨC XÂY DỰNG PRIORITY QUEUE

- Read more chapter 9

```
/** An abstract base class to assist implementations of the PriorityQueue interface.*/
public abstract class AbstractPriorityQueue<K,V>
                                        implements PriorityQueue<K,V> {
  //---------- nested PQEntry class ----------
  protected static class PQEntry<K,V> implements Entry<K,V> {
    private K k;    // key
    private V v;    // value
    public PQEntry(K key, V value) {
      k = key;
      v = value;
    }
    // methods of the Entry interface
    public K getKey() { return k; }
    public V getValue() { return v; }
    // utilities not exposed as part of the Entry interface
    protected void setKey(K key) { k = key; }
    protected void setValue(V value) { v = value; }
  } //---------- end of nested PQEntry class ----------

  // instance variable for an AbstractPriorityQueue
  /** The comparator defining the ordering of keys in the priority queue. */
  private Comparator<K> comp;
  /** Creates an empty priority queue using the given comparator to order keys. */
  protected AbstractPriorityQueue(Comparator<K> c) { comp = c; }
  /** Creates an empty priority queue based on the natural ordering of its keys. */
  protected AbstractPriorityQueue() { this(new DefaultComparator<K>()); }
  /** Method for comparing two entries according to key */
  protected int compare(Entry<K,V> a, Entry<K,V> b) {
    return comp.compare(a.getKey(), b.getKey());
  }
  /** Determines whether a key is valid. */
  protected boolean checkKey(K key) throws IllegalArgumentException {
    try {
      return (comp.compare(key,key) == 0);  // see if key can be compared to itself
    } catch (ClassCastException e) {
      throw new IllegalArgumentException("Incompatible key");
    }
  }
  /** Tests whether the priority queue is empty. */
  public boolean isEmpty() { return size() == 0; }
}
```

Code Fragment 9.5: The AbstractPriorityQueue class. This provides a nested PQEntry class that composes a key and a value into a single object, and support for managing a comparator. For convenience, we also provide an implementation of isEmpty based on a presumed size method.

Xây dựng PriorityQueue từ gói java.utils.*

- Xây dựng một lớp ProrityQueue kế thừa từ lớp TreeSet.
- Xây dựng các phương thức push() → add(Object o), pop() → first(), remove(), top() → first() dựa trên các phương thức đã có của TreeSet

```
public class PriorityQueue extends TreeSet{
public PriorityQueue (Comparator comparator)  {
super(comparator);
}
public PriorityQueue ()          {
super();
}

}
```

MÔ PHỎNG VIỆC XE QUA PHÀ TẠI MỘT BẾN PHÀ THEO QUI ĐỊNH CỦA LUẬT GIAO THÔNG ĐƯỜNG BỘ.

a) Khi đến bến phà, bến cầu phao, cầu treo các loại xe phải xếp hàng có trật tự đúng nơi quy định, không làm cản trở giao thông.

b) Khi xe qua phà mọi người trên xe phải xuống xe, trừ lái xe và những người mắc bệnh nặng không thể đi được.

c) Các loại xe cơ giới phải xuống phà trước rồi mới đến hành khách và xe cơ giới chỉ được lên bến khi hành khách lên bến hết (trừ loại phà có bố trí chở hành khách lên xuống phà và chỗ ngồi trên phà riêng biệt với các loại xe).

d) Những xe ưu tiên qua phà, cầu phao, cầu treo trước các loại xe khác theo thứ tự sau đây:

1- Các loại xe ghi ở điểm 1, 2, 3 khoản a của Điều 42 quyền ưu tiên khi

qua đường giao nhau.

2- Xe hộ đê khi có báo hiệu cấp 2 trở lên.

3- Đoàn xe có cảnh sát đi trước dẫn đường.

4- Đoàn xe tang.

5- Xe đi làm nhiệm vụ đảm bảo giao thông khẩn cấp ở phía trước.

6- Xe chở thư báo.

7- Xe phục vụ những yêu cầu đột xuất có tính chất khẩn cấp như cứu mùa màng, chống dịch... và xe chở thực phẩm tươi sống.

8- Xe chở khách công cộng.

Nếu các xe cùng một loạt xe ưu tiên như nhau đến bến phà, cầu phao cùng một lúc thì xe nào đến trước sẽ qua trước.

Thạc sỹ: Trần Lê Như Quỳnh

HƯỚNG DẪN CÁCH LÀM

- Tạo ra 1 lớp là phương tiện bên trong có ghi tên phương tiện, mã số xe, biển số xe, kích thước, trọng tải… là một Comparable hoặc Comparator.
- Tạo 1 lớp là Phà có thông tin về kích thước, tải trọng, mã phà.
- Tạo 1 lớp dùng để sắp xếp xe tại bến phà.