Implementing queue using stack

1)Enqueue

class MyQueue {

Stack<Integer> s1;

Stack<Integer> s2;

/\*\* Initialize your data structure here. \*/

public MyQueue()

{

s1=new Stack<Integer>();

s2=new Stack<Integer>();

}

/\*\* Push element x to the back of queue. \*/

public void push(int x)

{

while(!s2.isEmpty())

{

s1.push(s2.pop());

}

s2.push(x);

while(!s1.isEmpty())

{

s2.push(s1.pop());

}

}

/\*\* Removes the element from in front of queue and returns that element. \*/

public int pop()

{

return s2.pop();

}

/\*\* Get the front element. \*/

public int peek()

{

return s2.peek();

}

/\*\* Returns whether the queue is empty. \*/

public boolean empty()

{

return s2.isEmpty();

}

}

2)Dequeue

class MyQueue {

Stack<Integer> s1;

Stack<Integer> s2;

int front;

/\*\* Initialize your data structure here. \*/

public MyQueue()

{

s1=new Stack<Integer>();

s2=new Stack<Integer>();

}

/\*\* Push element x to the back of queue. \*/

public void push(int x)

{

if(s1.isEmpty())

{

front=x;

}

s1.push(x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

public int pop()

{

if(s2.isEmpty())

{

while(!s1.isEmpty())

{

s2.push(s1.pop());

}

}

return s2.pop();

}

/\*\* Get the front element. \*/

public int peek()

{

if(!s2.isEmpty())

{

return s2.peek();

}

return front;

}

/\*\* Returns whether the queue is empty. \*/

public boolean empty()

{

return s2.isEmpty() && s1.isEmpty();

}

}

/\*\*

\* Your MyQueue object will be instantiated and called as such:

\* MyQueue obj = new MyQueue();

\* obj.push(x);

\* int param\_2 = obj.pop();

\* int param\_3 = obj.peek();

\* boolean param\_4 = obj.empty();

\*/