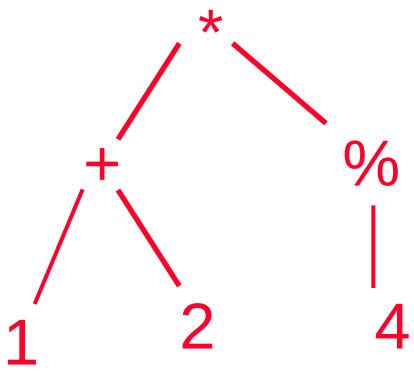
ADT's

- **Discussion of Assignment 6**
  - **Postfix form of expressions**
  - **Evaluation algorithm for postfix expressions**
- **Defined interface methods independent of chosen representation type.**
  - **e.g., a Queue with 2 Stacks as its representation**

Assignment 6

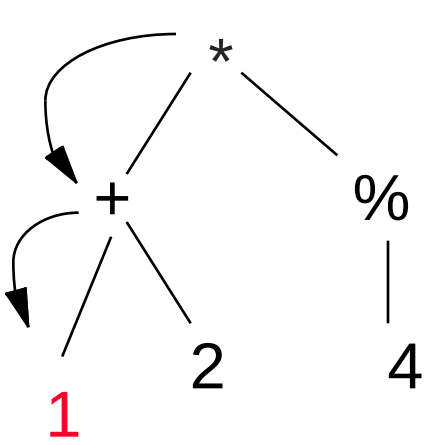
- **Postfix form of an expression is written with the operator last:**
  - **operand1 operand2 operator**
  - **No need for parentheses**
- **Examples**
  - **(1+2)*%4 is in postfix: 1 2 + 4 % * which is evaluated as: ((1 2 +) (4 %) *)**
  - **1 + 2 / 3 is in postfix 1 2 3 / + which is evaluated as: (1 (2 3 /) +)**
- **Postfix form can be read off the expression tree according to a specific traversal order**

Postfix Expression Evaluation

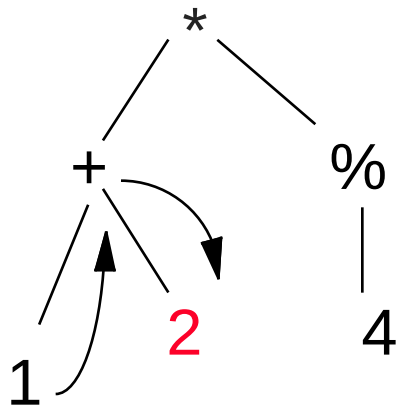


(1+2) * %4

Postfix expression results from a leftmost, depth-first traversal of tree, writing a node when you last visit it.

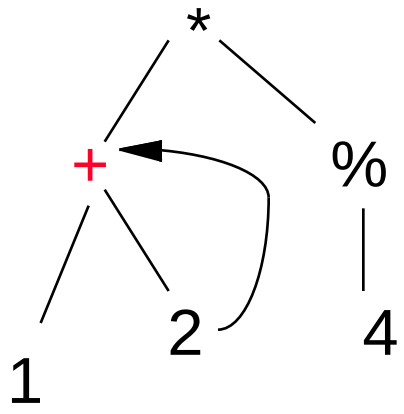


1

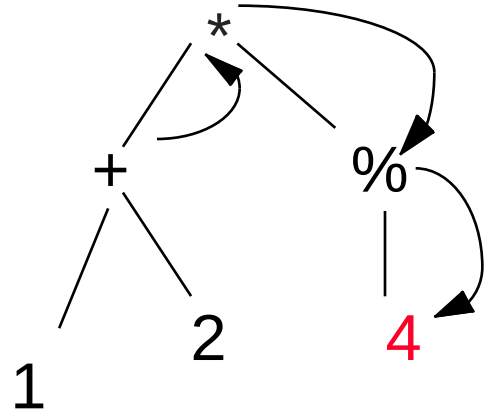


1 2

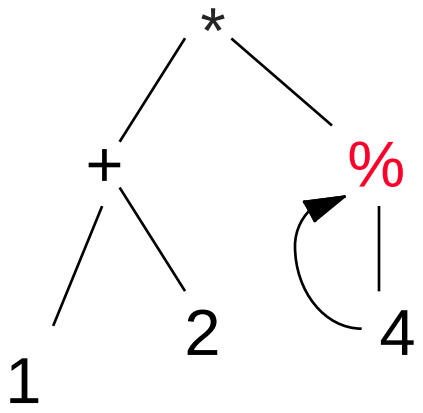
Assignment 6



1 2 +

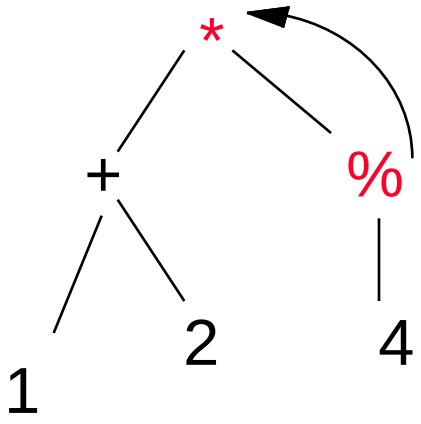


1 2 + 4



1 2 + 4 %

Assignment 6



This is called
a Postorder Traversal of a tree.

☺ More on trees in CS112

1 2 + 4 % *

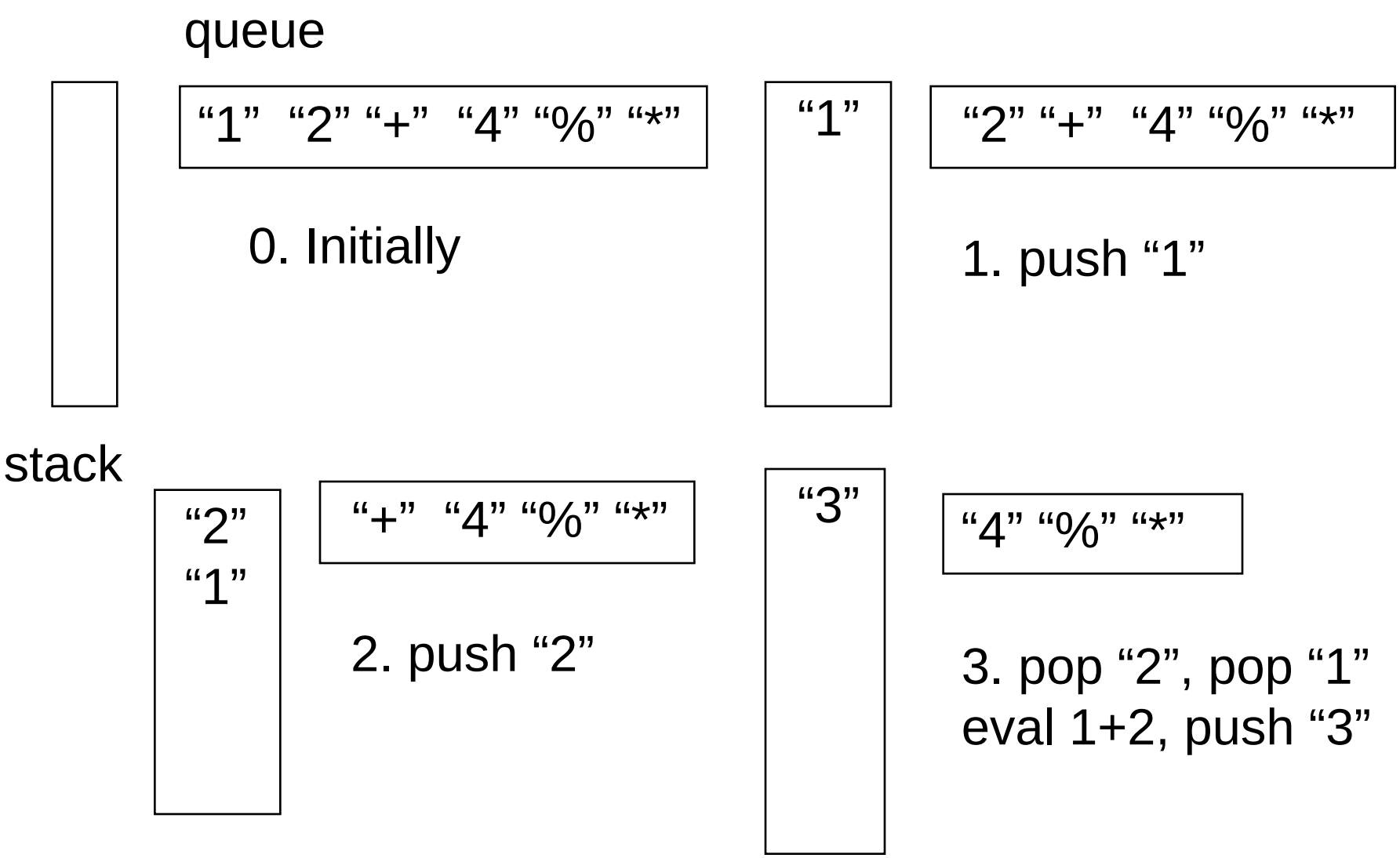
Final postfix form of expression obtained!

In the assignment, you are to create a Queue object filled with the String representations of ints and operators in the proper order.

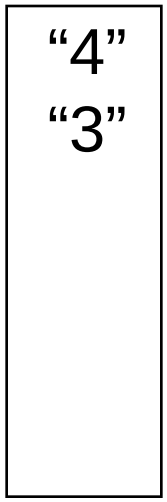
Evaluation of Postfix Form

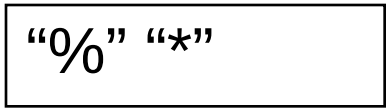
- **Simple evaluation algorithm uses a stack**
  - **Read the next item in the expression**
    - **If it is an operand, push it onto the stack**
    - **If it is an operator, determine how many operands it has, pop that many operands off the stack, evaluate the operator, push the value obtained onto the stack**
  - **When there are no more items in the expression, its value is on top of the stack**

Example

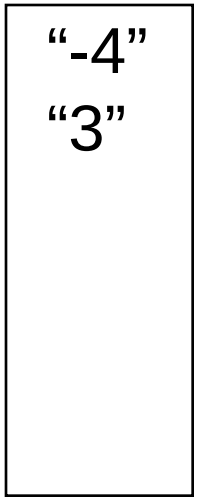

queue
"1" "2" "+" "4" "%" "*"
0. Initially
"1"
"2" "+" "4" "%" "*"
1. push "1"
stack
"2"
"1"
"+" "4" "%" "*"
2. push "2"
"3"
"4" "%" "*"
3. pop "2", pop "1"
eval 1+2, push "3"

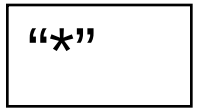
Example



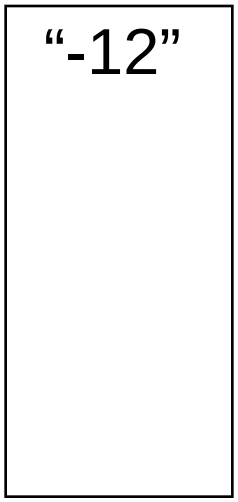


4. push “4”





5. pop “4”, evaluate %4, push -4.

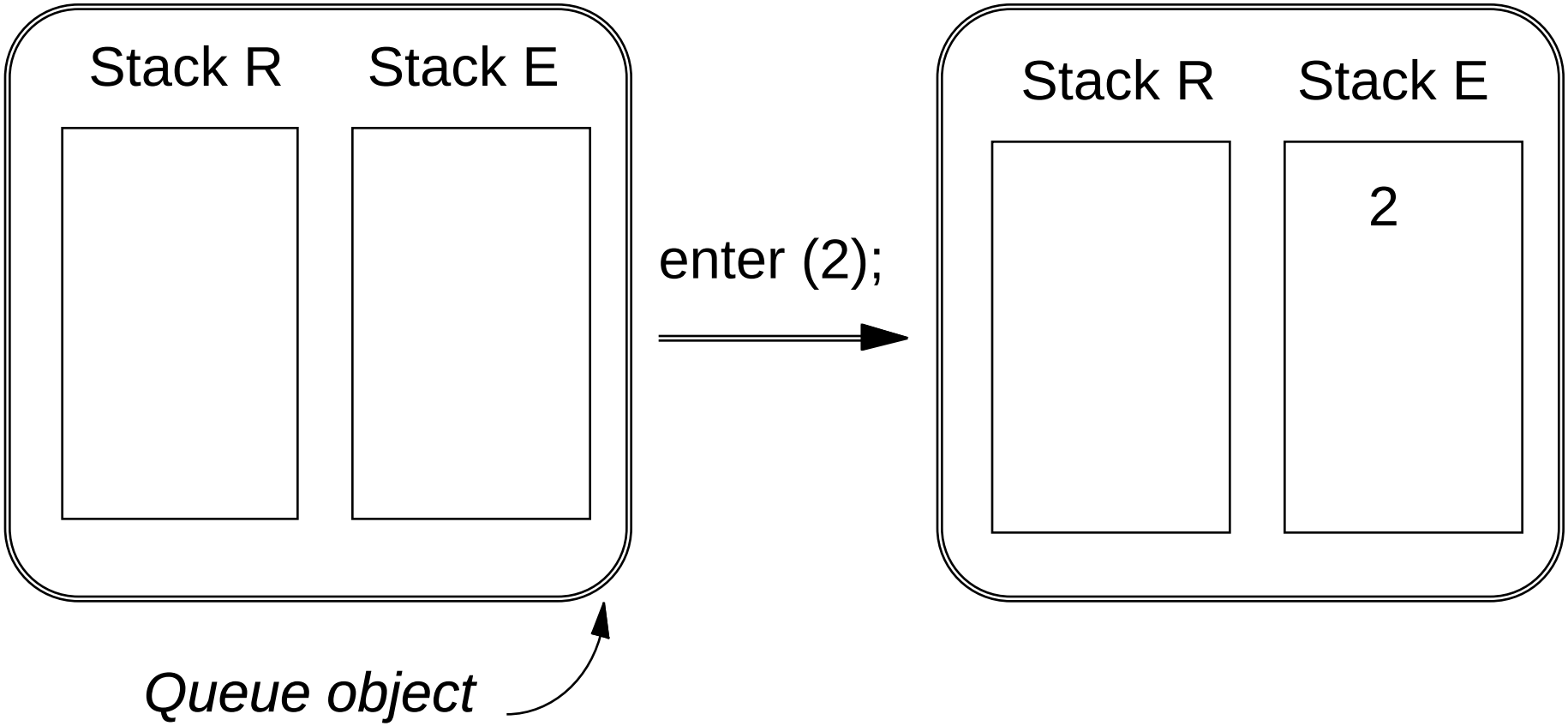


6. pop “-4”, pop “3”, evaluate -4*3, push “-12”.

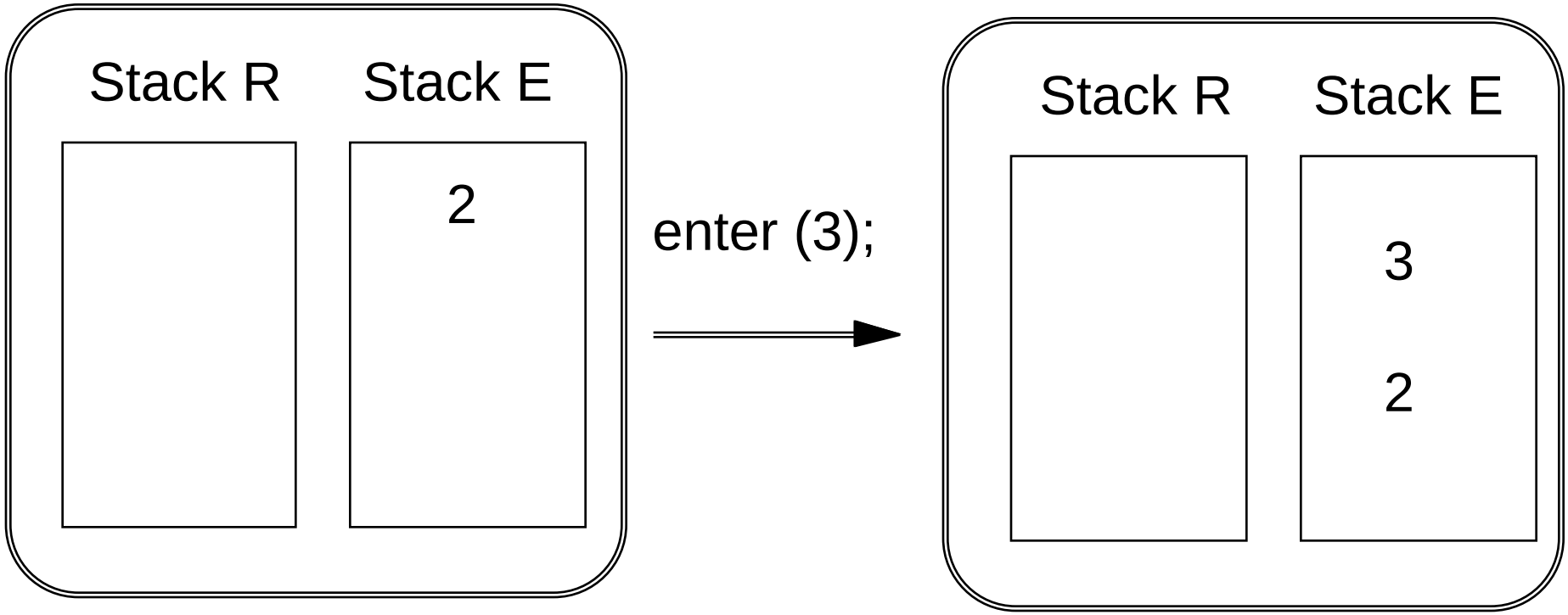
7. end of expression means value is -12.

Queues

- **If ADT is truly independent of representation type, we can change representation type *without* changing interface!**
- **Let's use 2 Stacks to represent a Queue object!**
  - **use Stack R for removes**
  - **use Stack E for enters**
  - **Have to rearrange things when Stack R is empty and a remove() is executed**

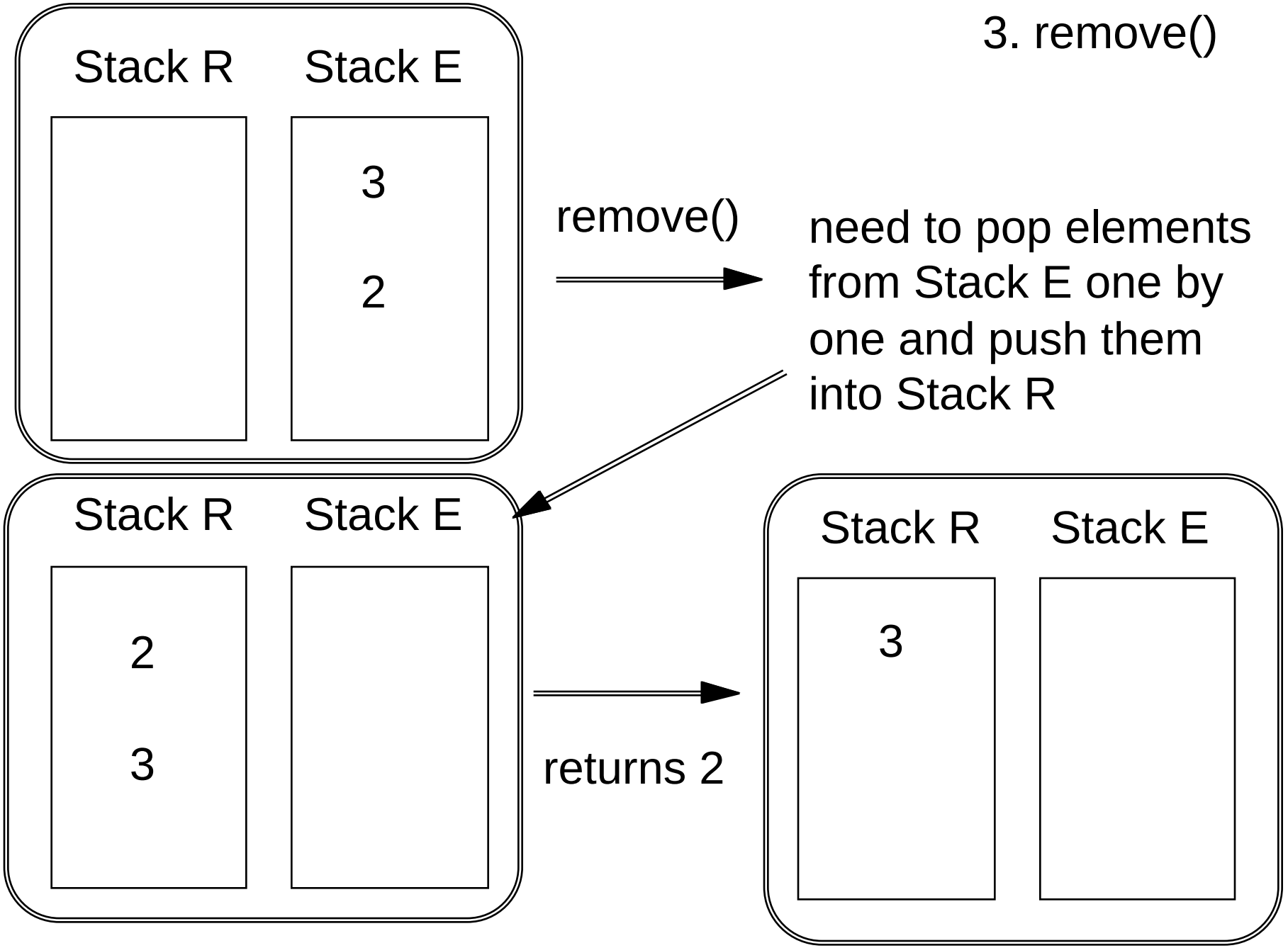


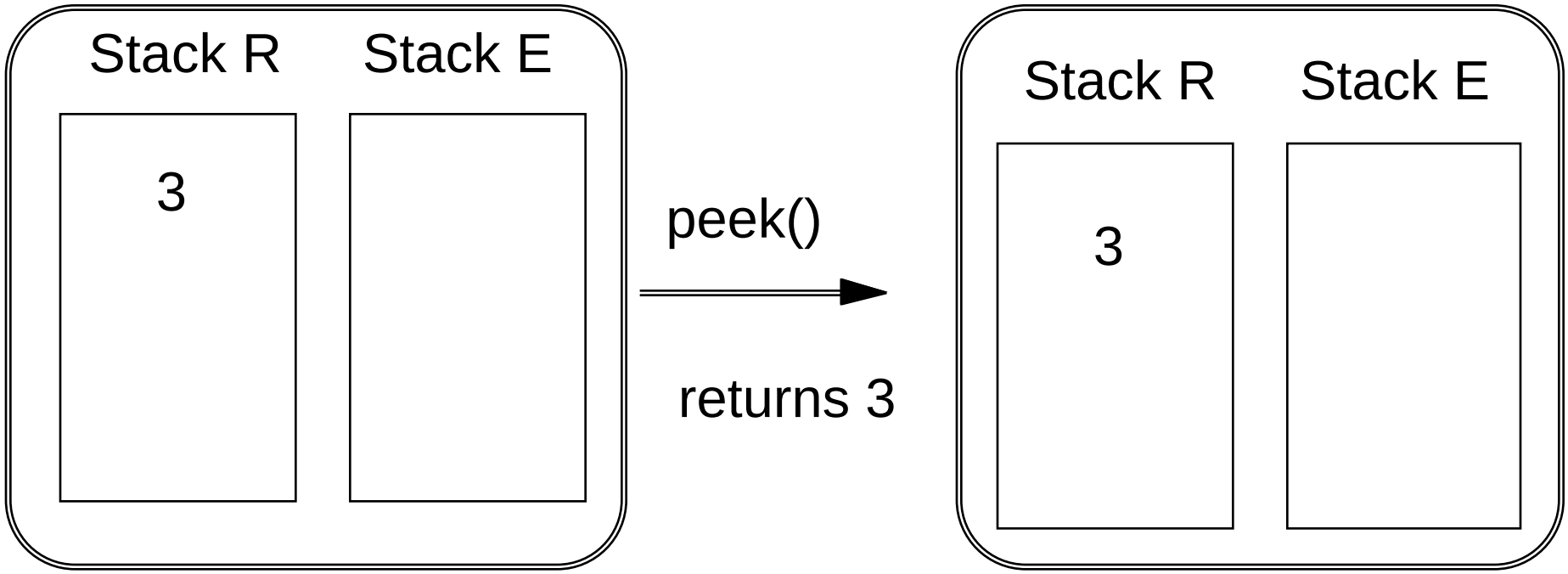
1. enter(2)



2. enter(3)

3. remove()


Stack R
Stack E
3
2
remove()
need to pop elements from Stack E one by one and push them into Stack R
Stack R
Stack E
2
3
returns 2
Stack R
Stack E
3



4. peek()

Note: peek() can be implemented using peek() in Stack class if stack R is not empty.

Further Considerations

- **Exceptions: remove() from or peek() on empty Queue corresponds to Stack R and Stack E both being empty**
- **Need no special case to handle adding to empty Queue and removing from Queue with only 1 element**
- **Need fixup step if do a remove() when Stack R is empty and Stack E isn't; push all of Stack E's elements 1 by 1 into Stack R; write auxiliary method for this job *refill()***

Queue with Stacks rep type

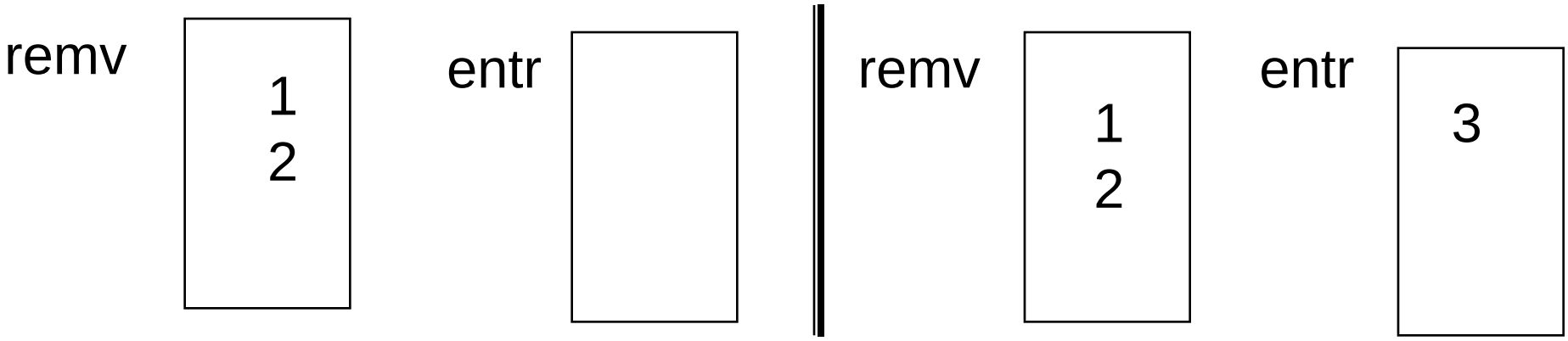
```
public class Queue extends Object {

  private Stack remv;//R stack
  private Stack entr;//E stack
  private int length;

  public Queue() {//empty queue is 2 empty
  stacks
    remv = new Stack();
    entr = new Stack();
    length = 0;
  }
```

not in the cs111.util.Queue class; instead look at /newstacks/Queue.java

enter Method

```
public void enter(Object newItem) {
    entr.push(newItem);//push newItem onto E stack
    length++;
  }
```

```
Queue q = new Queue();
Integer i = new Integer(3);
q.enter(i);//Why won't q.enter(3) work??
```



Private Auxiliary Methods

```
//returns true when entr stack can be emptied into
//remv stack in order to do a remove()
//
  private boolean bothEmpty(){
    if(remv.empty() && entr.empty()) return true;
    else return false;
  }

  private static void refill(Stack r1,Stack e2)
  throws StackException
  {//refills r1 from e2 in destructive manner
        while(!(e2.empty()))
        {
                Object o1 = e2.pop();
                r1.push(o1);
        }
  }
```

remove Method

```
public Object remove() throws QueueException,
  StackException
{   Object oo;
    if (this.bothEmpty()) throw new QueueException(
      "Attempt to remove from empty Queue");
    else {//check if need to reset remv before removal
          if(remv.empty()) refill(remv,entr);
          length--;
          oo = remv.pop();
          return oo;
  }
}
```

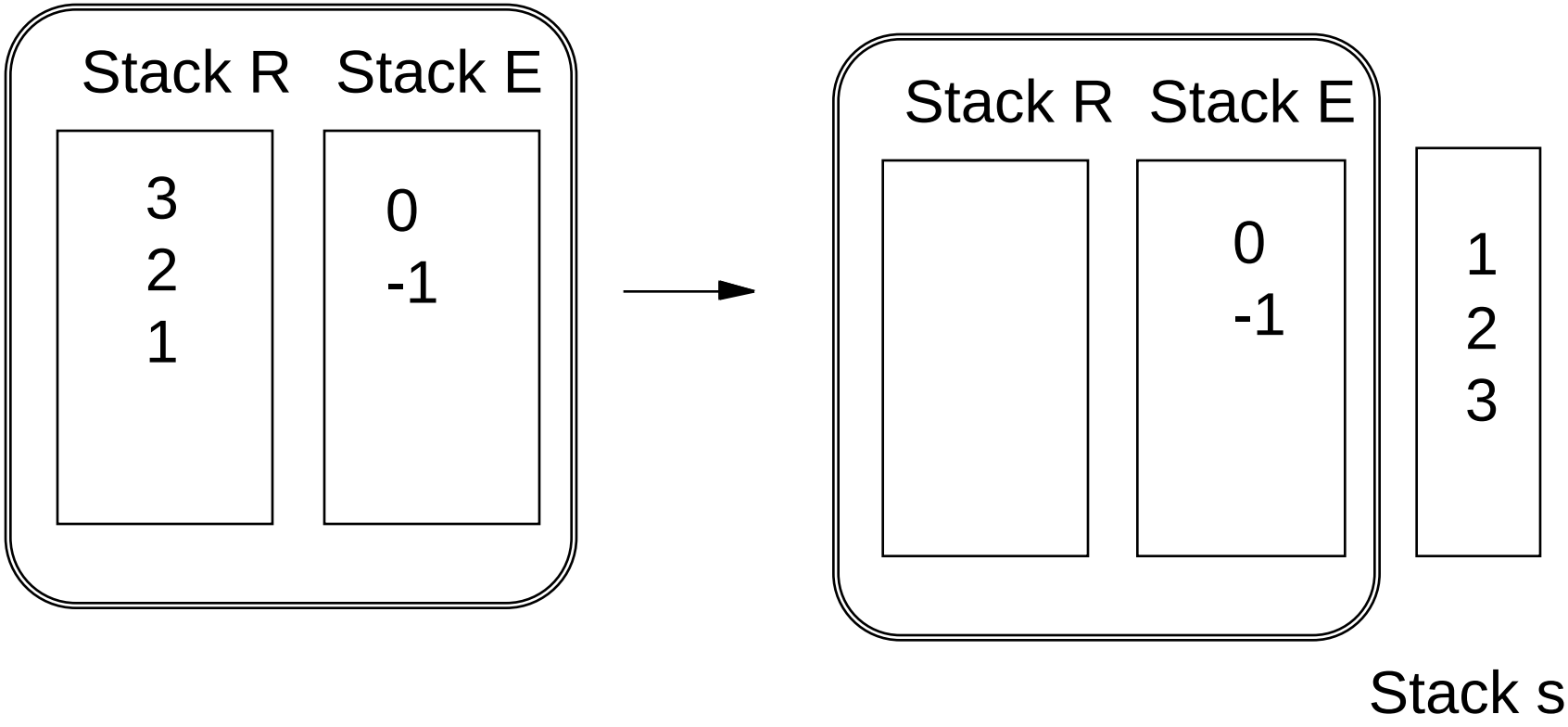
peek Method

```
public Object peek() throws QueueException,
  StackException
{
    Object oo;
    if (this.bothEmpty())
      throw new QueueException(
            "Attempt to peek at empty Stack");
    else    {if (remv.empty()) refill(remv,entr);
            oo = remv.peek();//peek() in Stack class
            return oo;
            }
}
```

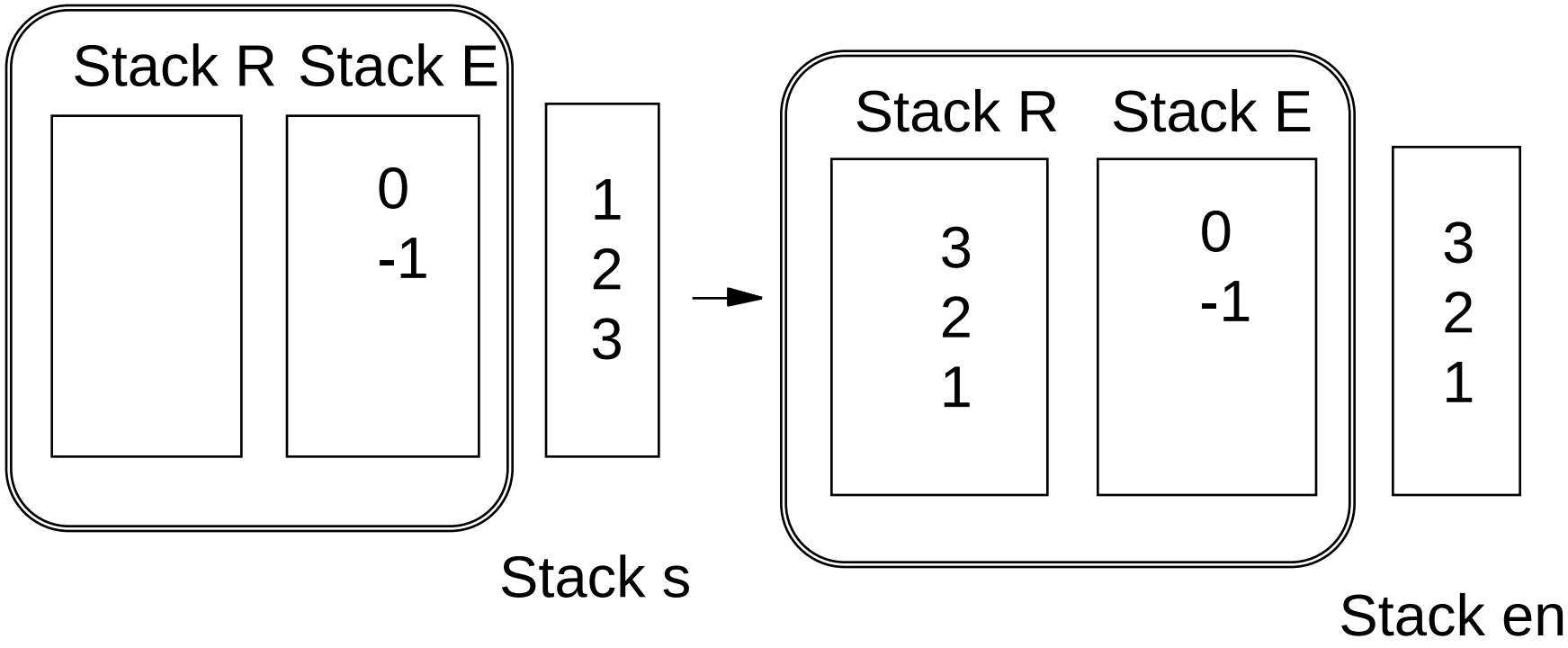
Enumeration - How to do it?

- **Say decide enumeration will feed elements first from Stack *remv* next from Stack *entr***
- **Need to get inside of *remv* and *entr* Stacks**
- **Need to save elements as pop them off, in order to rebuild *remv* and *entr* Stacks**
- **Combine both Stacks into one big Stack *en* and return StackEnumeration on *en* for getEnumeration() in Queue**

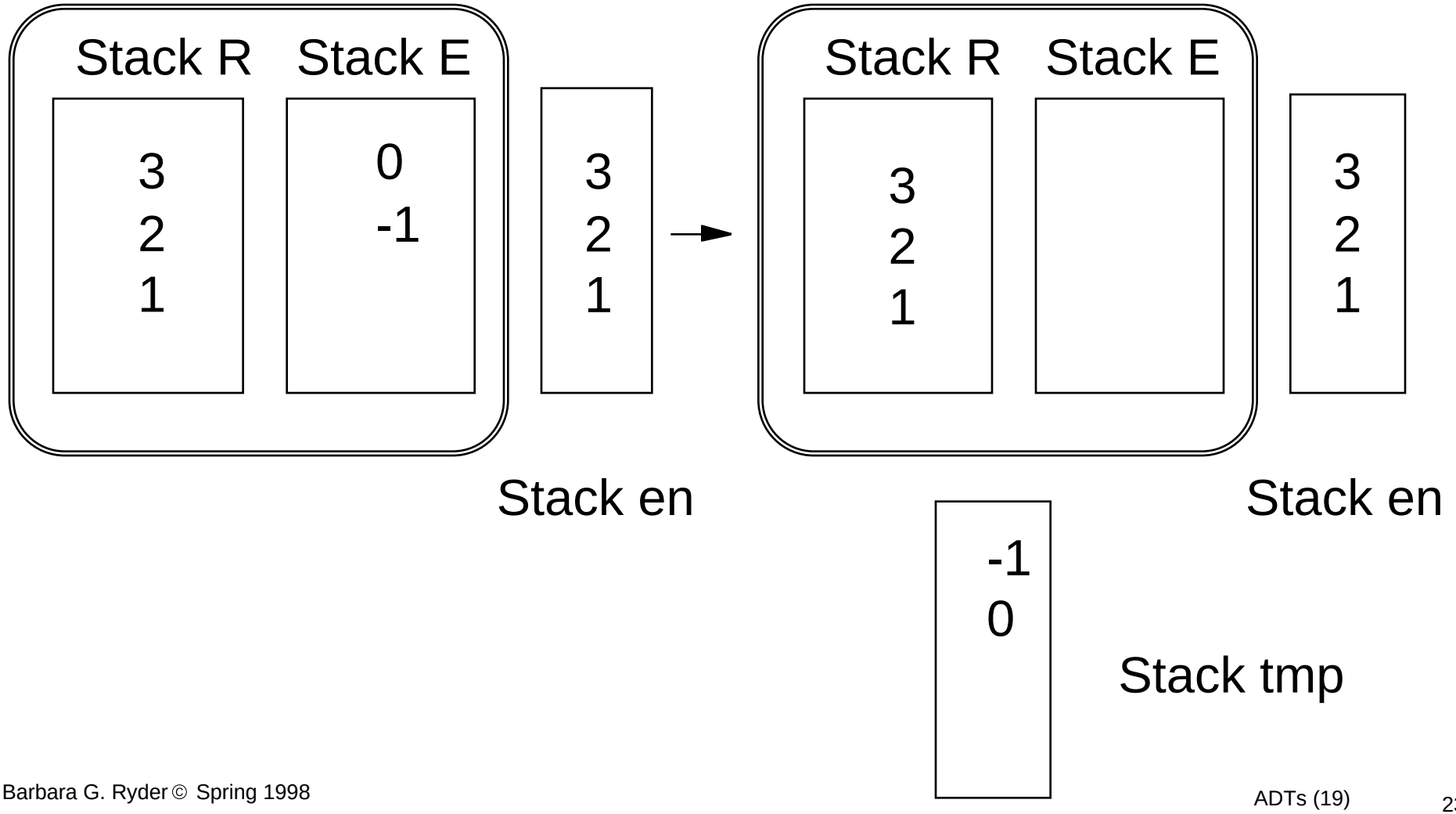
How to do it?



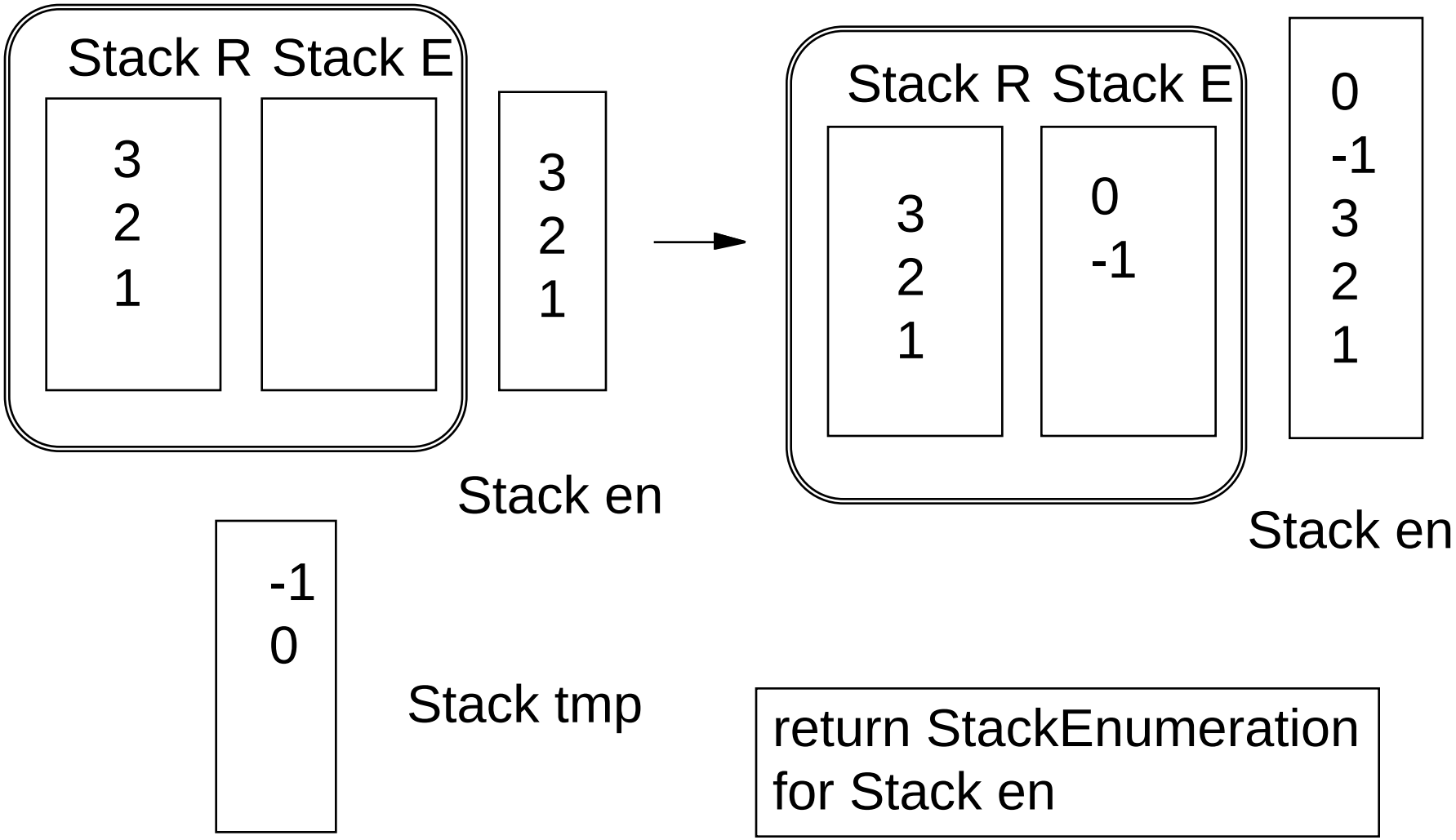
How to do it?


Stack R
Stack E
0
-1
1
2
3
Stack s
Stack R
Stack E
3
2
1
0
-1
3
2
1
Stack en

How to do it?


Stack R
Stack E
3
2
1
0
-1
3
2
1
Stack en
Stack R
Stack E
3
2
1
3
2
1
Stack en
-1
0
Stack tmp

How to do it?


Stack R
Stack E
3
2
1
3
2
1
Stack en
Stack R
Stack E
3
2
1
0
-1
0
-1
3
2
1
Stack en
-1
0
Stack tmp
return StackEnumeration
for Stack en

Enumeration Code (exerpt)

```
if (!(entr.empty())){
    while(!(entr.empty())){
          Object oo = entr.pop();
          en.push(oo);
          t.push(oo);
    }
    if (!(t.empty())) refill(w,t);
    entr = w;
}
return en.getEnumeration();
```

Key Points

- **Queue interface is the same although very different mechanism used to represent Queue objects**
- **User is oblivious to underlying rep type change**
- **Allows software updates for *running* programs**
- **Separates efficiency issues (manipulations of rep type) from ADT properties and operations**

Exerpt from Main in Queue

```
Queue q = new Queue();
Integer i = new Integer(4);
q.enter(i);
i = new Integer(6);
q.enter(i);
i = new Integer(-1);
q.enter(i);
 System.out.println("after enter 4,6,-1 " +
    q.toString());
 i = (Integer) q.remove();
 System.out.println("after remove " + q.toString()
    + " i= " + i.intValue());
 Integer j = (Integer) q.peek();
 System.out.println("after peek " + q.toString() +
    " j= " + j.intValue());
```

fill up Queue object

do actions on Queue

Exerpt from Main in Queue

```
Queue r = new Queue();
   r.enter("a");//note Strings are Objects!
   r.enter("b");
   r.enter("c");
   r.enter("d");
//test of first form of enumeration
   Enumeration e = r.getEnumeration();
   System.out.println("Print out queue r using
       getEnumeration\n");
   while (e.hasMoreElements())
   {  System.out.println((e.nextElement()).toString()
              + "\n");
   }
```

Output

```
after enter 4,6,-1 Queue length is 3
Queue is:
4
6
-1

after remove Queue length is 2
Queue is:
6
-1
 i= 4
after peek Queue length is 2
Queue is:
6
-1
 j= 6
Print out queue r using getEnumeration
a
b
c
d
7 remus!newstacks>
```