3 ΣΤΟΙΒΕΣ ΚΑΙ ΟΥΡΕΣ

3.1 ΣΤΟΙΒΕΣ

Στοίβα *(stack*) είναι μία λίστα στην οποία νέα στοιχεία μπορούν να προστεθούν και να αφαιρεθούν μόνο από τη μία άκρη της (**κορυφή** της στοίβας). Συχνά μία στοίβα αναφέρεται και σαν μία λίστα τύπου **LIFO** (**L**ast-**I**n-**F**irst-**O**ut), για να δηλώνεται έτσι ρητά η βασική της ιδιότητα, ότι το στοιχείο που θα προστεθεί τελευταίο στη στοιβάδα θα αφαιρεθεί πρώτο ή ισοδύναμα το πρώτο στοιχείο που θα προστεθεί στη στοιβάδα, αναγκαστικά πρέπει να αφαιρεθεί τελευταίο.

Βασικές πράξεις σε στοίβες:

Οι βασικές πράξεις (λειτουργίες) που ορίζονται για τον τύπο στοίβα αναφέρονται παρακάτω:

1) **Δημιουργία Στοίβας:** Με την πράξη αυτή δημιουργείται μία κενή στοίβα **S**, η οποία δεν περιέχει κανένα στοιχείο.

2) **Εισαγωγή στοιχείου σε Στοίβα - push(P):** Δοθείσης μίας στοίβας **S**, το στοιχείο **P** τοποθετείται στην κορυφή της στοίβας **S**, σαν τελευταίο στοιχείο.

3) **Εξαγωγή στοιχείου από Στοίβα - pop( ):** Το στοιχείο **P** που βρίσκεται στην κορυφή της στοίβας **S** αφαιρείται από αυτήν και επιστρέφεται.

4) **Ελεγχος κενής στοιβάδας - isEmpty( ):** Δοθείσης μίας στοίβας **S**, επιστρέφει την τιμή **true,** εάν η στοίβα **S** δεν έχει κανένα στοιχείο, ενώ στην αντίθετη περίπτωση την τιμή **false.**

Επιπλέον των πράξεων αυτών που θεωρούνται βασικές συνήθως ορίζονται και οι πράξεις:

5) **Κορυφή της στοίβας - top( ):** Το στοιχείο **P** που βρίσκεται στην κορυφή της στοίβας **S** επιστρέφεται, χωρίς όμως να αφαιρεθεί από αυτήν.

6) **Μέγεθος της στοίβας - size( ):** Δοθείσης μίας στοίβας **S** επιστρέφει το πλήθος των στοιχείων της (ενεργό μήκος της στοίβας).

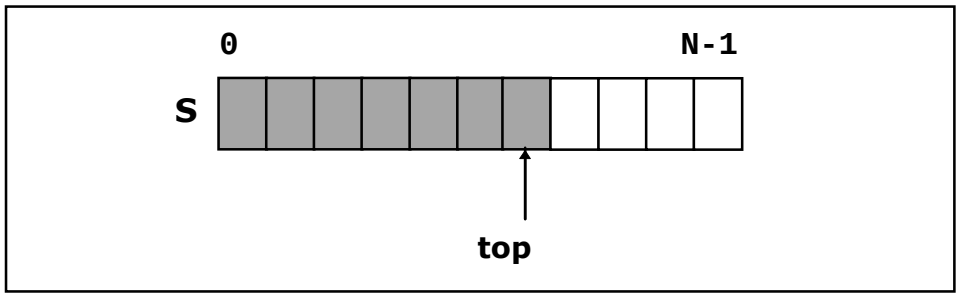
Η παρακάτω **διασύνδεση** (***interface***) ορίζει **τον αφηρημένο τύπο δεδομένων (*abstract data type*)** στοίβα:

```
public interface Stack
{
public int size();
    // επιστρέφει το μέγεθος της στοίβας

public boolean isEmpty();
    // αληθεύει εάν η στοίβα είναι κενή

public Object top( ) throws StackEmptyException;
    // επιστρέφει το στοιχείο που βρίσκεται στην κορυφή της στοίβας

public void push(Object item) throws StackFullException;
    // εισάγει ένα νέο στοιχείο στην κορυφή της στοίβας

public Object pop( ) throws StackEmptyException;
    // εξάγει και επιστρέφει το στοιχείο που βρίσκεται στην κορυφή της στοίβας
}
```

Τμήμα Κώδικα 1

3.1.1 Υλοποίηση Στοίβας με τη Βοήθεια Πίνακα

Μία στοίβα μπορεί να αναπαρασταθεί με τη βοήθεια ενός πίνακα (συγκεκριμένου μήκους) και ενός ακεραίου "ενδείκτη" ο οποίος δείχνει το στοιχείο του πίνακα που αποτελεί την κορυφή της στοίβας.

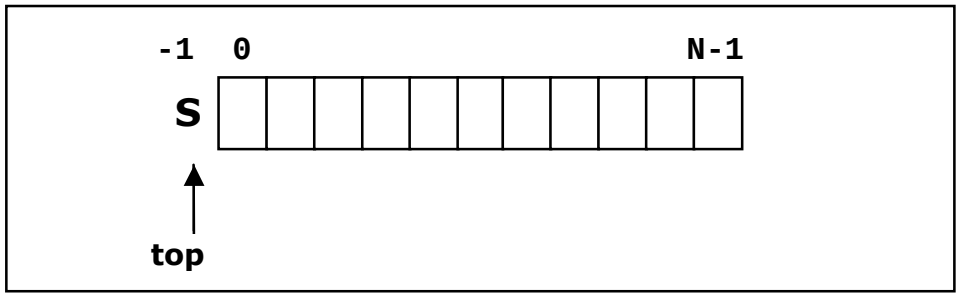


Ο πίνακας αυτός μαζί με τον ενδείκτη μπορούν να ομαδοποιηθούν με τη βοήθεια δηλώσεων της κλάσης **ArrayStack**, που υλοποιεί τον αφηρημένο τύπο **Stack**, όπως φαίνεται παρακάτω:

```
public class ArrayStack implements Stack
{
      public static final int CAPACITY = 1000;
      private int capacity;
      private Object[ ] S;
      private int top = -1;
      ..........................
}
```

1) Δημιουργία Στοίβας:

Η κενή στοίβα που δημιουργείται μπορεί να παρασταθεί με έναν "άδειο" πίνακα και την αρχική τιμή του ενδείκτη **top** = -1



Το μέγεθος της στοίβας μπορεί να καθορίζεται είτε έμμεσα π.χ 1000 στοιχεία είτε άμεσα κατά τη δημιουργία της στοίβας. Οι δομητές της κλάσης ArrayStack για τις δύο αυτές περιπτώσεις είναι οι εξής:

```
public ArrayStack( ) {
    this(CAPACITY);
}

public ArrayStack(int cap) {
    capacity = cap;
    S = new Object[capacity];
}
```

2) Εισαγωγή στοιχείου στη Στοίβα:

Η διαδικασία εισαγωγής ενός στοιχείου στη στοίβα συνίσταται στην αύξηση του ενδείκτη **top** κατά ένα και την τοποθέτηση του στοιχείου στη θέση του πίνακα που δείχνει ο **top**. Στην περίπτωση που ο χώρος της στοίβας έχει εξαντληθεί από προηγούμενες εισαγωγές στοιχείων, δημιουργείται κατάσταση εξαίρεσης (υπερχείλιση στοίβας). Η μέθοδος **push** της ***ArrayStack*** είναι η εξής:

```
public void push(Object item) {
    if (size( )== capacity)
        throw new StackFullException("Stack overflow");
    S[++top] = item;
}
```

Η εξαίρεση **StackFullException** πρέπει να οριστεί με τη βοήθεια ξεχωριστής κλάσης, η οποία επεκτείνει την κλάση RuntimeException.

3) Εξαγωγή στοιχείου από Στοίβα:

Η διαδικασία εξαγωγής ενός στοιχείου από τη στοίβα συνίσταται στην επιστροφή του στοιχείου που βρίσκεται στην κορυφή της στοίβας και τη μείωση του ενδείκτη **top** κατά ένα. Εάν κατά την πράξη εξαγωγής στοιχείου η στοίβα βρεθεί άδεια δημιουργείται κατάσταση εξαίρεσης (άδεια στοίβα).

```
public Object pop( ) throws StackEmptyException {
    Object element;
    if (isEmpty( ))
        throw new StackEmptyException("Stack is empty");
    element = S[top];
    S[top--] = null;  //!!! Χρειάζεται για τον garbage collector
    return element;
}
```

Η εξαίρεση **StackEmptyException** πρέπει επίσης να οριστεί με τη βοήθεια ξεχωριστής κλάσης.

4) Ελεγχος κενής στοίβας:

Είτε στην περίπτωση που η στοίβα έχει μόλις δημιουργηθεί είτε στην περίπτωση που η στοίβα δεν έχει κανένα στοιχείο μετά από διαδοχικές εισαγωγές και εξαγωγές στοιχείων ο ενδείκτης top θα πρέπει να έχει την τιμή -1:

```
public boolean isEmpty( ) {
    return (top < 0);
}
```

Στο τμήμα κώδικα 2 δίνεται ολόκληρη η υλοποίηση της κλάσης ArrayStack και στο τμήμα κώδικα 3 των κλάσεων που υλοποιούν τις εξαιρέσεις.

```
public class ArrayStack implements Stack
{
    public static final int CAPACITY = 1000;
    private int capacity;
    private Object[ ] S;
    private int top = -1;

    public ArrayStack( ) {
        this(CAPACITY);
}
```

```
    public ArrayStack(int cap) {
        capacity = cap;
        S = new Object[capacity];
    }

    public int size( ) {
        return (top+1);
    }

    public boolean isEmpty( ) {
        return (top < 0);
    }

    public void push(Object item) {
        if (size( )== capacity)
            throw new StackFullException("Stack overflow");
        S[++top] = item;
    }

    public Object top( ) throws StackEmptyException {
        if (isEmpty( ))
            throw new StackEmptyException("Stack is empty");
        return S[top];
    }

    public Object pop() throws StackEmptyException {
        Object element;
        if (isEmpty())
            throw new StackEmptyException("Stack is empty");
        element = S[top];
        S[top--] = null; //!!! για τον garbage collector
        return element;
    }
}
```

Τμήμα Κώδικα 2

```
public class StackEmptyException extends RuntimeException
{
      public StackEmptyException(String err)
      { super(err); }
}

public class StackFullException extends RuntimeException
{
      public StackFullException(String err)
      { super(err); }
}
```

Τμήμα Κώδικα 3

Σαν ένα παράδειγμα χρήσης της στοίβας στο τμήμα κώδικα 4 δίνεται η κλάση **ReverseArray** η οποία υλοποιεί τη διαδικασία αντιστροφής των στοιχείων ενός πίνακα χρησιμοποιώντας τις βασικές πράξεις της στοίβας.

```
import java.io.*;

class ReverseArray
{
      public static String[ ] Reverse(String[ ] a)
      {
          int i;
          ArrayStack s = new ArrayStack( );
          for (i=0; i<a.length; ++i)
              s.push(a[i]);
          i=0;

          while (!s.isEmpty( ))
          {    a[i]=(String)s.pop();
              i++;
          }
          return a;
      }
```

```
    public static void main(String[ ] args) throws IOException
    {
        String b[ ] = {"nikos","demos","maria","lina","thomas"};
        String c[ ] = new String[b.length];
        ReverseArray RA = new ReverseArray();
        c = RA.Reverse(b);
        for (int i=0; i<c.length; ++i)
            System.out.println(c[i]);
    }
}
```

Τμήμα Κώδικα 4

3.2 ΟΥΡΕΣ

Ουρά (queque) είναι μια λίστα στην οποία μπορούν να προστεθούν στοιχεία μόνο στη μία άκρη (πίσω) και να αφαιρεθούν μόνο από την άλλη (μπροστά). Μια ουρά αναφέρεται και σαν λίστα τύπου **FIFO** (**F**irst-**I**n- **F**irst-**O**ut).

Βασικές πράξεις σε ουρές:
Οι βασικές πράξεις (λειτουργίες) που ορίζονται για τον τύπο ουρά είναι οι εξής:

1) **Δημιουργία Ουράς:** Με την πράξη αυτή δημιουργείται μία κενή ουρά **Q**, η οποία δεν περιέχει κανένα στοιχείο.

2) **Εισαγωγή στοιχείου σε Ουρά - enqueue(P):** Δοθείσης μίας ουράς **Q**, το στοιχείο **P** τοποθετείται στο πίσω μέρος της σαν τελευταίο στοιχείο.

3) **Εξαγωγή στοιχείου από Ουρά - dequeue(P):** Δοθείσης μίας ουράς **Q**, το στοιχείο **P** που βρίσκεται στην αρχή της αφαιρείται από αυτήν και επιστρέφεται.

4) **Ελεγχος κενής Ουράς - isEmpty( ):** Επιστρέφει την τιμή **true**, εάν η ουρά **Q** είναι κενή, ενώ στην αντίθετη περίπτωση την τιμή **false**.

Επιπλέον των πράξεων αυτών που θεωρούνται βασικές συνήθως ορίζεται και η πράξη:

5) **Μέγεθος της ουράς - size( ):** Δοθείσης μίας ουράς **Q** επιστρέφει το πλήθος των στοιχείων της (ενεργό μήκος της ουράς).

6) **Πρώτο στοιχείο της ουράς - front( ):** Δοθείσης μίας ουράς **Q** επιστρέφει το πρώτο στοιχείο της ουράς, χωρίς όμως να το αφαιρεί από αυτήν.

Η παρακάτω **διασύνδεση** (***interface***) ορίζει τον **αφηρημένο τύπο δεδομένων (*abstract data type*) ουρά** (τμήμα κώδικα 5). Στο τμήμα κώδικα 6 δίνονται οι κλάσεις που υλοποιούν τις εξαιρέσεις που σχετίζονται με την ουρά:

```
public interface Queue
{
public int size( );
    // επιστρέφει το μέγεθος (αριθμός στοιχείων) της ουράς

public boolean isEmpty( );
    // αληθεύει εάν η ουρά είναι κενή

public Object front( ) throws QueueEmptyException;
    // επιστρέφει το στοιχείο που βρίσκεται στο εμπρός μέρος της ουράς

public void enqueue(Object item);
    // εισάγει ένα νέο στοιχείο στο πίσω μέρος της ουράς

public Object dequeue( ) throws QueueEmptyException;
    // εξάγει και επιστρέφει το στοιχείο που βρίσκεται
    // στο εμπρός μέρος της ουράς
}
```

Τμήμα κώδικα 5

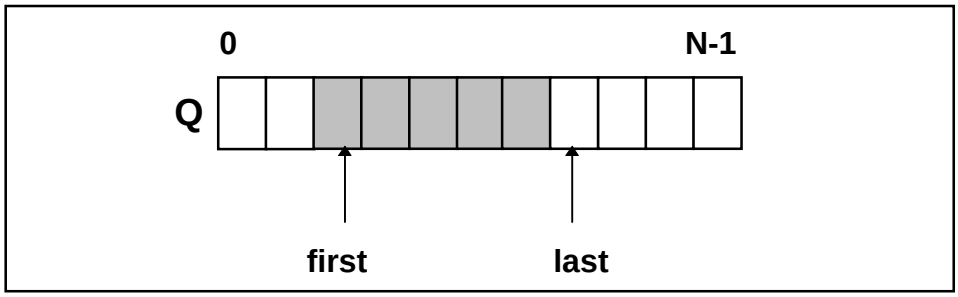
```
public class QueueEmptyException extends RuntimeException
{
      public QueuekEmptyException(String err)
      { super(err); }
}
```

```
public class QueueFullException extends RuntimeException
{
      public QueueFullException(String err)
      { super(err); }
}
```

Τμήμα Κώδικα 6

3.2.1 Υλοποίηση Ουράς με τη Βοήθεια Πίνακα

Μια ουρά μπορεί να αναπαρασταθεί με τη βοήθεια ενός πίνακα και δύο ακεραίων ενδεικτών από τους οποίους ο ένας δείχνει το στοιχείο του πίνακα που αποτελεί το τελευταίο στοιχείο της ουράς και ο άλλος το στοιχείο του πίνακα που αποτελεί το πρώτο στοιχείο της.



Ο πίνακας αυτός μαζί με τους δύο ενδείκτες μπορούν να ομαδοποιηθούν με τη βοήθεια δηλώσεων της κλάσης **ArrayQueue**, που υλοποιεί τον αφηρημένο τύπο **Queue**, όπως φαίνεται παρακάτω:

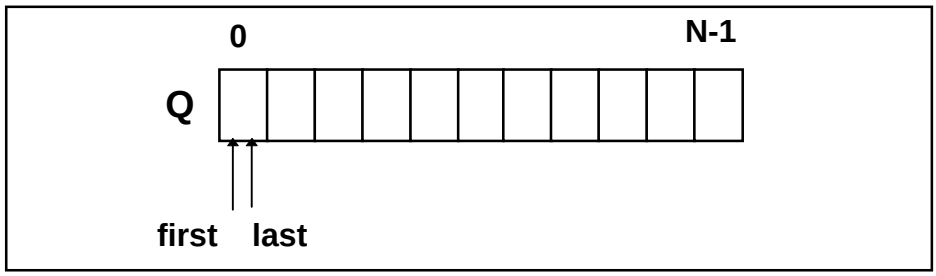
```
public class ArrayQueue implements Queue
{
      public static final int CAPACITY = 1000;
      private int capacity;
      private Object[ ] Q;
      private int first = 0;
      private int last = 0;
      ..............
```

1) Δημιουργία *Ουράς*:

Το μέγεθος της ουράς μπορεί να καθορίζεται είτε έμμεσα π.χ 1000 στοιχεία είτε άμεσα κατά τη δημιουργία της. Οι δομητές της κλάσης ArrayQueue για τις δύο αυτές περιπτώσεις είναι οι εξής:

```
public ArrayQueue() {
    this(CAPACITY);
}

public ArrayQueue(int cap) {
    capacity = cap;
    Q = new Object[capacity];
}
```

Κατά την αρχική δημιουργία της ουράς οι τιμές των ενδεικτών **first** και **last** είναι 0.



2) Εισαγωγή Στοιχείου στην *Ουρά*:

Η διαδικασία εισαγωγής ενός στοιχείου στην ουρά συνίσταται στην τοποθέτηση του στοιχείου στη θέση του πίνακα που δείχνει ο ενδέικτης **last** και στη συνέχεια την αύξηση του ενδείκτη κατά ένα. Στην περίπτωση που ο χώρος της ουράς έχει εξαντληθεί από προηγούμενες εισαγωγές στοιχείων, δημιουργείται κατάσταση εξαίρεσης (υπερχείλιση ουράς). Η μέθοδος **enqueue** της **ArrayQueue** είναι η εξής:

```
public void enqueue(Object item) {
    if (last == capacity)
        throw new QueueFullException("Queue overflow");
    Q[last++] = item;
}
```

Η εξαίρεση **QueueFullException** έχει οριστεί με τη βοήθεια ξεχωριστής κλάσης, η οποία επεκτείνει την κλάση RuntimeException.

Πρέπει να σημειωθεί ότι κατά κανόνα η κατάσταση της υπερχείλισης της ουράς είναι "εικονική", με την έννοια ότι ο πίνακας διαθέτει άδεια στοιχεία, τα οποία με βάση τη συγκεκριμένη υλοποίηση της **enqueue** δεν μπορούν να χρησιμοποιηθούν. Υπάρχουν διάφοροι τρόποι αντιμετώπισης της εικονικής υπερχείλισης της ουράς, ένας από τους οποίους δίνεται στην παράγραφο 3.2.2.

3) Εξαγωγή Στοιχείου από την Ουρά:

Η διαδικασία εξαγωγής ενός στοιχείου από την ουρά συνίσταται στην επιστροφή του στοιχείου που βρίσκεται στη θέση **first** του πίνακα και στη συνέχεια την αύξηση του ενδείκτη κατά ένα. Στην περίπτωση που η ουρά είναι άδεια, δημιουργείται κατάσταση εξαίρεσης (άδεια ουρά). Η μέθοδος **dequeue** της **ArrayQueue** είναι η εξής:

```
public Object dequeue( ) throws QueueEmptyException {
    Object item;
    if (isEmpty( ))
        throw new QueueEmptyException("Queue is empty");
    item = Q[first];
    Q[first++] = null; //!!! για τον garbage collector
    return item;
}
```

Η εξαίρεση **QueueEmptyException** έχει επίσης οριστεί με τη βοήθεια ξεχωριστής κλάσης.

4) Ελεγχος κενής ουράς:

Είτε στην περίπτωση που η ουρά έχει μόλις δημιουργηθεί είτε στην περίπτωση που η ουρά δεν έχει κανένα στοιχείο μετά από διαδοχικές εισαγωγές και εξαγωγές στοιχείων οι ενδείκτες **first** και **last** θα πρέπει να έχουν την ίδια τιμή:

```
public boolean isEmpty( ) {
    return (first==last);
}
```

Στο τμήμα κώδικα 7 δίνεται ολόκληρη η υλοποίηση της κλάσης ArrayQueue.

```
public class ArrayQueue implements Queue
{
	public static final int CAPACITY = 1000;
	private int capacity;
	private Object[ ] Q;
	private int first = 0;
	private int last = 0;

	public ArrayQueue() {
		this(CAPACITY);
	}

	public ArrayQueue(int cap) {
		capacity = cap;
		Q = new Object[capacity];
	}

	public int size( ) {
		return (last-first);
	}
	public boolean isEmpty( ) {
		return (first==last);
	}

	public void enqueue(Object item) {
		if (last == capacity)
			throw new QueueFullException("Queue overflow");
		Q[last++] = item;
	}

	public Object front( ) throws QueueEmptyException {
		if (isEmpty( ))
			throw new QueueEmptyException("Queue is empty");
		return Q[first];
	}
```

```
    public Object dequeue( ) throws QueueEmptyException {
        Object item;
        if (isEmpty( ))
            throw new QueueEmptyException("Queue is empty");
        item = Q[first];
        Q[first++] = null; //!!! για τον garbage collector
        return item;
    }
}
```

Τμήμα Κώδικα 7

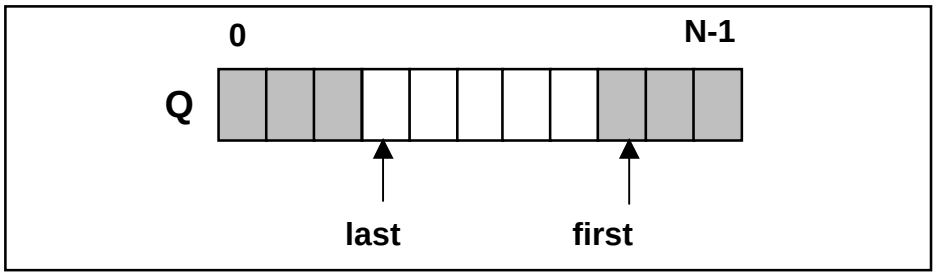
Σαν παράδειγμα χρήσης της ουράς στο τμήμα κώδικα 8 φαίνεται η χρήση μιάς ουράς για τη δημιουργία αντιγράφου ενός πίνακα χαρακτήρων.

```
import java.io.*;
public class QueueTest
{
    public static String[ ] QCopy(String[ ] a)   {
        int i;
        ArrayQueue q = new ArrayQueue();
        String[] aa =new String[a.length];
        for (i=0; i<a.length; ++i)
            q.enqueue(a[i]);
        i=0;
        while (!q.isEmpty( ))
        {   aa[i]=(String)q.dequeue( );
            i++;
        }
        return aa;  }

    public static void main(String[ ] args) throws IOException {
        String[ ] b = {"nikos","demos","maria","lina","thomas"};
        String[ ] c = new String[b.length];
        QueueTest QA = new QueueTest();
        c = QA.QCopy(b);
        for (int i=0; i<c.length; ++i)
            System.out.println(c[i]);    }
}
```

Τμήμα Κώδικα 8

3.2.2 Κυκλική Ουρά

Όπως αναφέρθηκε στην παράγραφο 3.2.1 η υλοποίηση της ουράς με τη βοήθεια ενός πίνακα μπορεί να οδηγήσει στο φαινόμενο της εικονικής υπερχείλισης. Ας θεωρήσουμε για παράδειγμα ότι έχουμε εισάγει και στη συνέχεια εξάγει ένα στοιχείο **N-1**φορές στην ουρά. Στην περίπτωση αυτή οι ενδείκτες **first** και **last** θα έχουν την τιμή **N** και η επόμενη προσπάθεια εισαγωγής στοιχείου στην ουρά θα οδηγηθεί σε αποτυχία, παρά το γεγονός ότι ο πίνακας που φιλοξενεί τα στοιχεία είναι εντελώς άδειος! Λύση στο πρόβλημα αυτό αποτελεί η κυκλική ουρά. Στην περίπτωση αυτή θεωρούμε ότι το επόμενο στοιχείο του **Q[N-1]** στον πίνακα είναι το **Q[0]** εφόσον αυτό είναι κενό. Για να γίνει δυνατή η υλοποίηση της κυκλικής ουράς πρέπει στις πράξεις εισαγωγής και εξαγωγής στοιχείου να τροποποιήσουμε την αύξηση των ενδεικτών με **(first+1)moduloN και (last+1)moduloN** αντίστοιχα, όπου ο τελεστής **modulo** αντιστοιχεί στο υπόλοιπο της ακεραίας διαίρεσης (στη Java χρησιμοποιείται ο τελεστής %). Μία εικόνα της κυκλικής ουράς μετά από έναν αριθμό πράξεων εισαγωγής - εξαγωγής φαίνεται στο παρακάτω σχήμα:



Στο τμήμα κώδικα 9 δίνεται η κλάση **SArrayQueue** που υλοποιεί τον τύπο Queue σαν μία κυκλική ουρά.

```
public class SArrayQueue implements Queue
      // Υλοποίηση κυκλικής ουράς
{
      public static final int CAPACITY = 1000;
      private int capacity;
      private Object[ ] Q;
      private int first = 0;
      private int last = 0;

      public SArrayQueue( ) {
          this(CAPACITY);
      }

      public SArrayQueue(int cap) {
          capacity = cap;
          Q = new Object[capacity];
      }

      public int size( ) {
          return (capacity-first+last)%capacity;
      }

      public boolean isEmpty( ) {
          return (first==last);
      }

      public void enqueue(Object item) {
          if (size( ) == capacity)
              throw new QueueFullException("Queue overflow");
          Q[last] = item;
          last = (last+1)%capacity;
      }

      public Object front( ) throws QueueEmptyException {
          if (isEmpty())
              throw new QueueEmptyException("Queue is empty");
          return Q[first];
      }
```

```
    public Object dequeue( ) throws QueueEmptyException {
        Object item;
        if (isEmpty( ))
            throw new QueueEmptyException("Queue is empty");
        item = Q[first];
        Q[first] = null; //!!! για τον garbage collector
        first=(first+1)%capacity;
        return item;
    }
}
```

Τμήμα Κώδικα 9

3.3 ΑΣΚΗΣΕΙΣ

1. Δώστε το περιεχόμενο της στοίβας (stack) μετά την εκτέλεση της παρακάτω σειράς από πράξεις:

Π Α Ρ * Α * * Π Ο Λ * * * Υ Ε * * * Υ * Κ Ο * Λ Ο *

Στην περίπτωση αυτή ένα γράμμα ερμηνεύεται σαν πράξη εισαγωγής (του γράμματος) στη στοίβα, ενώ ο χαρακτήρας "*" ερμηνεύεται σαν πράξη εξαγωγής (ενός γράμματος) από τη στοίβα.

2. Αν η εκτέλεση της σειράς των πράξεων της άσκησης 1 γινόταν σε μία ουρά (queue), ποιό θα ήταν το τελικό της περιεχόμενο;

3. Υλοποιήστε τη δομή δεδομένων στοίβα (stack) με τη χρήση της κλάσης Vector της Java.

4. Μία αριθμητική παράσταση περιέχει παρενθέσεις. Να γραφεί ένα πρόγραμμα το οποίο να ελέγχει εάν οι παρενθέσεις αυτές έχουν χρησιμοποιηθεί σωστά. Να χρησιμοποιηθεί η δομή δεδομένων στοίβα.

5. Περιγράψτε πως είναι δυνατή η υλοποίηση μίας στοίβας με τη βοήθεια δύο ουρών.

6. Υλοποιήστε τη δομή δεδομένων ουρά (queue) με τη χρήση της κλάσης Vector της Java.

7. Περιγράψτε πως είναι δυνατή η υλοποίηση μίας ουράς με τη βοήθεια δύο στοιβών.

8. Στην περίπτωση της κυκλικής ουράς που παρουσιάστηκε στην παράγραφο 3.2.2 θεωρήστε ότι έχουμε εισάγει **N** στοιχεία, χωρίς να έχουμε εξάγει κανένα. Θα έχουμε τότε **first = last** όπως ακριβώς και στην περίπτωση που η ουρά είναι άδεια. Πως μπορούμε να αντιμετωπίσουμε αυτό το πρόβλημα;