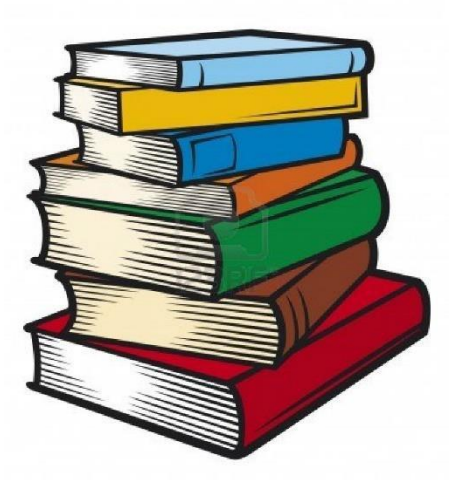
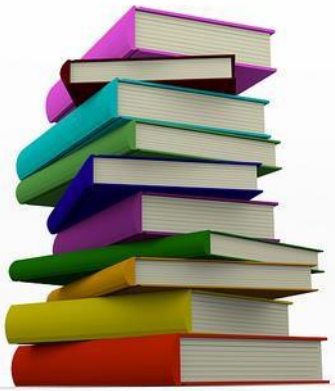
# Stacks



# Apstraktni tipovi podataka (ADT)

Def.: Apstraktni tip podataka (ADT) je skup objekata sa skupom operacija.

- Može se posmatrati kao matematička apstrakcija
- ADT definicija uključuje:
  - podatke, operacije, uslove i greške.
  - implementacije podrazumevaju algoritme zajedno sa svojim vremenom i prostorom složenosti

**Primeri ADT:**

- Liste, stacks, redovi
- Setovi
- Stabla, grafovi

**Operacije:**

- Add, insert, push, enqueuer, remove, delete, dequeue, pop, contains, union, find, itd.

# Stack ADT

- Stack ADT skladišti proizvoljno objekte
- Umetanja i brisanja prate last-in first-out šemu
- Glavne stack operacije:
  - push(object): umeće element
  - object pop(): uklanja i vraća poslednji umetnuti element
- Pomoćne stack operacije:
  - object top(): daje poslednji umetnuti element bez njegovog uklanjanja
  - integer size(): daje broj uskladištenih elemenata
  - boolean isEmpty(): označava da nije ni jedan elementi uskladišten

# Izuzetci

- Pokušaj izvršavanja operacije ADT-a ponekad može izazvati stanje greške, koje se naziva exception
- Exception kažu da su “thrown” operacijom koja se ne može izvršiti
- U Stack ADT, operacije pop i top nije moguće izvršiti ako je stack prazan
- Pokušaj izvršenja pop ili top na praznom stacu throws EmptyStackException

# Aplikacije Stacka

- Direktne aplikacije
  - Page-visited history u Web browser
  - Undo sequence u text editoru
  - Chain of method calls in the Java Virtual Machine
  - Recursion stack
  - Search algoritmi
- Indirektne aplikacije
  - Pomoćne data structure za algoritme
  - Komponente drugih struktura podataka

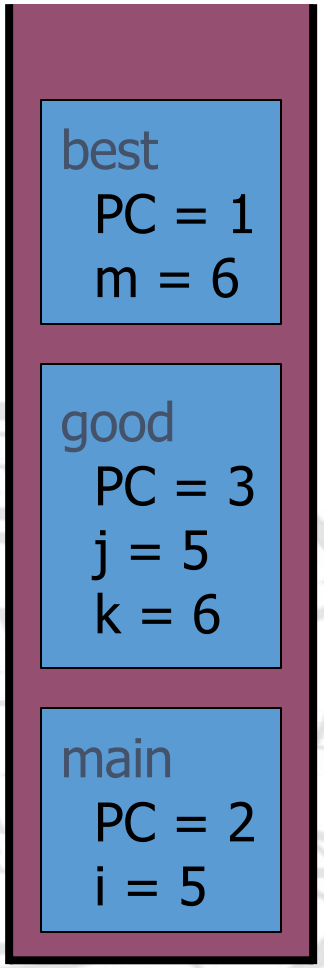
# Primer: Method Stack u JVM

- Java Virtual Machine (JVM) vodi evidenciju o lancu aktivnih metoda sa stackom
- Kada je metod pozvan, JVM gura na steku okvir koji sadrži
  - Lokalne promenljive i povratnu vrednost
  - Brojač programa, prateći izraz koji se izvršava
- Kada se metod završi, njegov okvir iskače iz stacka a kontrola se prosleđuju metodi na vrhu stacka
- Omogućava **rekurziju**

```
main() {
  int i = 5;
  good(i);
  }

good(int j) {
  int k;
  k = j+1;
  best(k);
  }

best(int m) {
  …
  }
```

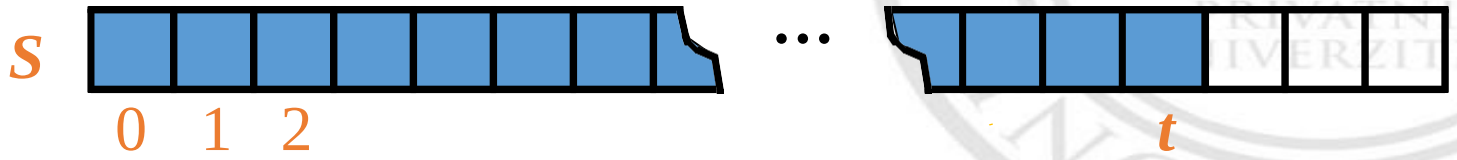


# Array-bazirani Stack

- Jednostavan način implementacije Stack ADT koristi niz
- Dodajemo elemente sleva nadesno
- Promenljiva prati indeks od top elementa

```
Algorithm size()
  return t + 1

Algorithm pop()
  if isEmpty() then
    throw EmptyStackException
  else
    t ← t − 1
    return S[t + 1]
```



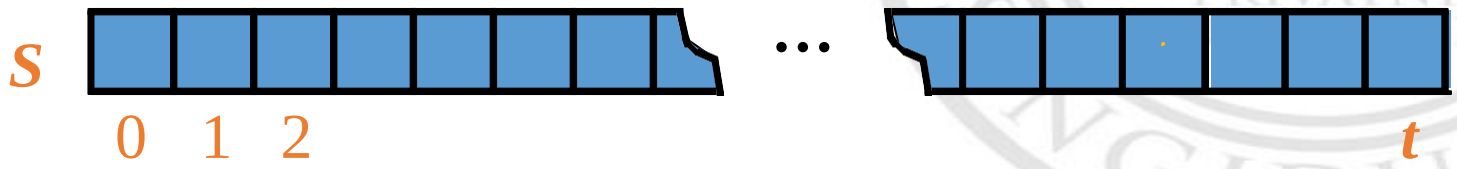
# Array-bazirani Stack (cont.)

- Niz koji skladišti stek elemente može postati pun
- Operacija guranja će onda baciti FullStackException
  - Ograničenje implementacije zasnovane na nizu
  - Nije suštinski za stack ADT

| **Algorithm** *push*(*o*) | # Operations |
|---|---|
| **if** $t = S.length - 1$ **then** | 2 |
| **throw** *FullStackException* | 1 |
| **else** | |
| $t \leftarrow t + 1$ | 2 |
| $S[t] \leftarrow o$ | 2 |
| | 6 |

Take Max: T(n) = 6

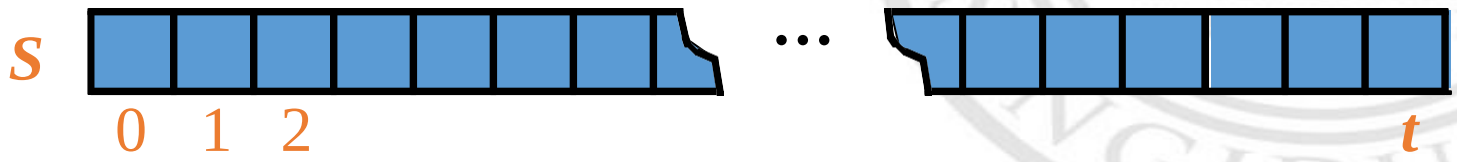
T(n) is O(1)



# Array-bazirani Stack (cont.)

- Pored toga, možda ćemo želeti da znamo sadržaj vrha niza
- Ali ne želimo da uklonimo element na vrhu

```
Algorithm top()
  if isEmpty() then
    throw EmptyStackException
  return S[t]
```



# Performanse i ograničenja

- Performanse
  - Neka $n$ bude broj elemenata u nizu
  - Prostor koji se koristi je $O(n)$
  - Svaka operacija se pokreće u $O(1)$ vremenu
- Ograničenja
  - Maksimalna veličina steka mora biti definisana apriori i ne može se menjati
  - Pokušaj guranja novog elementa u pun niz uzrokuje izuzetak specifičan za implementaciju
  - Ova situacija se može rešiti pomoću niza koji se može proširiti

# Ostale aplikacije Stacka

- Podudaranje zagrade (matching brackets)
- HTML tag matchings
- Reversing an array
- Čitanje infix i postfix artitmetičkih izraza
- Procena aritmetičkih izraza
- Context-free jezici i parseri
- Rekurzivni stack u kompajleru
- Search algoritmi
- Computing spans

# Bracket Matching

- Svaka “(”, “{”, ili “[” mora biti uparen sa podudarajućom “)”, “}”, ili “]” respektivno
  - ispravan: ()(( ))({([ ])})
  - ispravan: ((()(( ))({([ ])}))
  - netačno: )(( ))({([ ])}
  - netačno: ({[ ])} (upareni parovi ne smeju da se preklapaju međusobno)
  - netačno: (

# Bracket Matching Algoritam

**Algorithm** BracketMatch(*X,n*):

***Input:*** An array *X* of *n* tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

***Output:*** **true** if and only if all the grouping symbols in *X* match Let *S* be an empty stack

**for** *i*=0 to *n*-1 **do**

**if** *X*[*i*] is an opening grouping symbol **then**

*S*.push(*X*[*i*])

**else if** *X*[*i*] is a closing grouping symbol **then**

**if** *S*.isEmpty() **then**

**return false** *{ništa sa čem bi se poklopilo}*

**if** *S*.pop() does not match the type of *X*[*i*] **then return false** *{pogrešan tip}*

**if** *S*.isEmpty() **then**

**return true** *{svaki simbol se podudara}*

**else**

**return false** *{neki simboli se nikada nisu poklapali}*

# Bracket Matching Primer

**Algorithm** BracketMatch(*X*,*n*):

***Input:*** An array *X* of *n* tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

***Output:*** **true** if and only if all the grouping symbols in *X* match

Let *S* be an empty stack

**for** *i*=0 to *n*-1 **do**

  **if** *X*[*i*] is an opening grouping symbol **then**

    *S*.push(*X*[*i*])

  **else if** *X*[*i*] is a closing grouping symbol **then**

    **if** *S*.isEmpty() **then**

      **return false** *{nothing to match with}*

    **if** *S*.pop() does not match the type of *X*[*i*] **then**

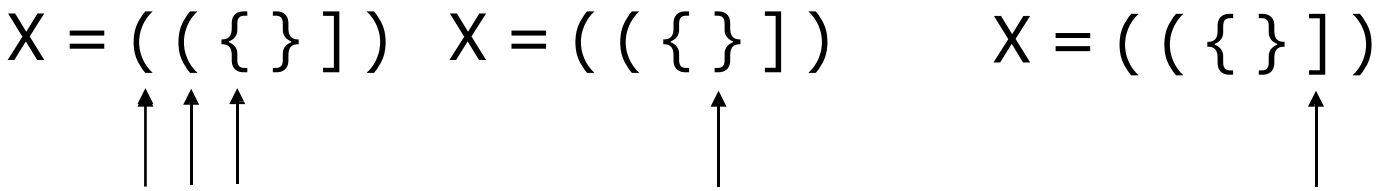
      **return false** *{wrong type}*

**if** *S*.isEmpty() **then**

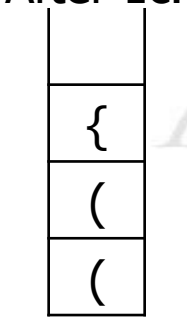
  **return true** *{every symbol matched}*

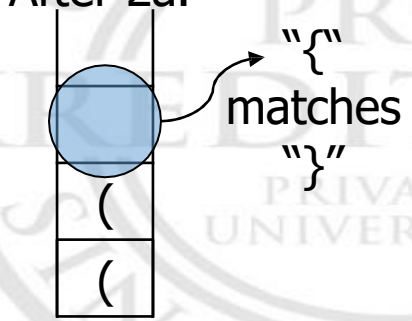
**else**

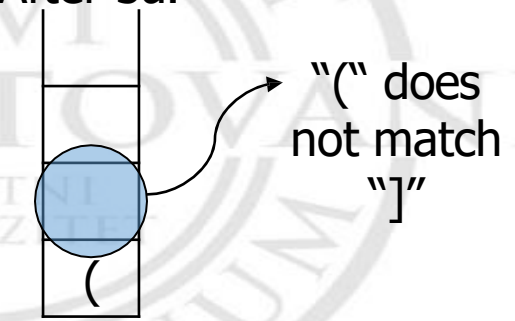
  **return false** *{some symbols were never matched}*




1. Opening brackets are pushed
   a. S.push "("
   b. S.push "("
   c. S.push "{"
2. Closing brackets are checked
   a. S.pop "{"
   b. "{" matches "}"
3. Closing brackets are checked
   a. S.pop "("
   b. "(" does not match "]"



# Još primera

**Algorithm** BracketMatch(*X,n*):

***Input:*** An array *X* of *n* tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

***Output:*** **true** if and only if all the grouping symbols in *X* match

Let *S* be an empty stack

**for** *i*=0 to *n*-1 **do**

  **if** *X*[*i*] is an opening grouping symbol **then**

    *S*.push(*X*[*i*])

  **else if** *X*[*i*] is a closing grouping symbol **then**

    **if** *S*.isEmpty() **then**

      **return false** *{nothing to match with}*

    **if** *S*.pop() does not match the type of *X*[*i*] **then**

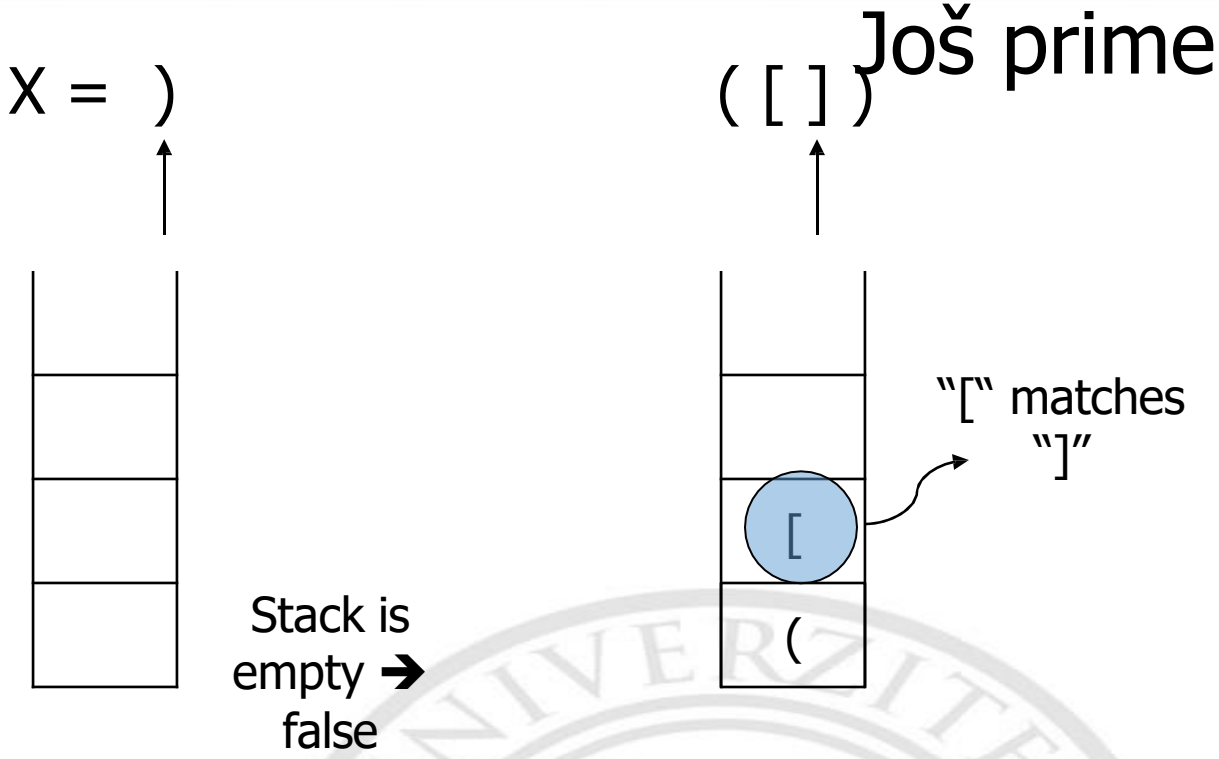
      **return false** *{wrong type}*

**if** *S*.isEmpty() **then**

  **return true** *{every symbol matched}*

**else**

  **return false** *{some symbols were never matched}*



# Izbalansirane zagrade bez Steka?

- Podudaranje zagrada se ne može rešiti bez korišćenja stacka!

Fact 1:

- Dokazano je da je jezik uravnoteženih zagrada nije regularan (ref [4])
- Videt primer:

https://www.ics.uci.edu/~goodrich/teach/cs162/hw/HW3Sols.pdf

- Kao takav, ne može se prepoznati od strane konačanih automata
- Mora se koristiti pushdown automat, koji se rešava na stacku

Fact 2:

- U praksi, neki pokušaji su urađeni da bi se to rešilo bez upotrebe stacka
- Na primer, pogledajte ref. [5]
- Međutim, rešenje je rekurzivno i, u izvršavanju, rekurzija koristi stack

# HTML Tag Matching

For correct HTML code, each <name> should pair with a matching </name>

```
<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>
```

## The Little Boat

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.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

# HTML Tag Matching Algoritam

**Algorithm** HTMLTagMatch(*X,n*):

***Input:*** An array *X* of *n* tokens, each of which is either an HTML tag or a text character

***Output:*** **true** if and only if all the HTML tags in *X* match

Let *S* be an empty stack

**for** *i*=0 to *n*-1 **do** *{n je broj tokena u X}*

    **if** *X*[*i*] is an opening HTML tag **then**

        *S*.push(*X*[*i*])

    **else if** *X*[*i*] is a closing HTML tag **then**

        **if** *S*.isEmpty() **then**

            **return false** *{ništa sa čim bi se poklopilo}*

        **if** *S*.pop() does not match the tag of *X*[*i*] **then**

            **return false** *{pogrešna oznaka}*

**if** *S*.isEmpty() **then**

    **return true** *{svaka oznaka se podudara}*

**else**

    **return false** *{neke oznake se nikada nisu podudarile}*

# HTML Tag Matching - Example

**Algorithm** HTMLTagMatch(*X,n*):
***Input:*** An array *X* of *n* tokens, each of which is either an HTML tag or a text character
***Output:*** **true** if and only if all the HTML tags in *X* match

Let *S* be an empty stack
**for** *i*=0 to *n*-1 **do** *{n is the number of tokens in X}*
  **if** *X*[*i*] is an opening HTML tag **then**
    *S*.push(*X*[*i*])
  **else if** *X*[*i*] is a closing HTML tag **then**
    **if** *S*.isEmpty() **then**
      **return false** *{nothing to match with}*
    **if** *S*.pop() does not match the tag of *X*[*i*] **then**
      **return false** *{wrong tag}*
**if** *S*.isEmpty() **then**
  **return true** *{every tag matched}*
**else**
  **return false** *{some tags were never matched}*

```
<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>
```

# Tag Matching Implementacija u Javi

It is similar to bracket matching:

```
import java.util.StringTokenizer;
import datastructures.Stack;
import datastructures.NodeStack;
import java.io.*;
/** Simplified test of matching tags in an HTML document. */
public class HTML { /** Nested class to store simple HTML tags */
        public static class Tag { String name; // The name of this tag
                boolean opening; // Is true i. this is an opening tag
                public Tag() { // Default constructor
                        name = "";
                        opening = false;
                }
                public Tag(String nm, boolean op) { // Preferred constructor
                name = nm;
                opening = op;
                }
                /** Is this an opening tag? */
                public boolean isOpening() { return opening; }
                /** Return the name of this tag */
                public String getName() {return name; }
        }
        /** Test if every opening tag has a matching closing tag. */
        public boolean isHTMLMatched(Tag[ ] tag) {
                Stack S = new NodeStack(); // Stack for matching tags
                for (int i=0; (i<tag.length) && (tag[i] != null); i++) {
                        if (tag[i].isOpening())
                        S.push(tag[i].getName()); // opening tag; push its name on the stack
                        else {
                             if (S.isEmpty()) // nothing to match
                                        return false;
                             if (!((String) S.pop()).equals(tag[i].getName())) // wrong match
                                        return false;
                        }
                }
                if (S.isEmpty())
                     return true; // we matched everything
                return false; // we have some tags that  were never matched
        }
```

# Tag Matching Implementacija (cont'd)

```
public final static int  CAPACITY = 1000;       // Tag array size upper bound
  /* Parse an HTML document into an array of html tags */
  public  Tag[ ]  parseHTML(BufferedReader r) throws  IOException {
                String line;       // a line of text
                boolean          inTag =  false   ;                  // true iff we are in a tag
                 Tag[ ]  tag  =  new  Tag[CAPACITY];  // our tag array (initially all null)
                int  count   =  0                    ;                  // tag counter
                while  ((line  =  r.readLine()) !=  null)  {
            // Create a string tokenizer for HTML tags (use  <  and  >  as delimiters)
            StringTokenizer st  =  new  StringTokenizer(line,"<> \t",true);
            while  (st.hasMoreTokens())  {
                    String token  = (String)  st.nextToken();
                    if  (token.equals("<"))  // opening a new HTML tag
                            inTag  =  true;
                    else if  (token.equals(">"))  // ending an HTML tag
                            inTag  =  false;
                    else if  (inTag)  { // we have a opening or closing HTML tag
                            if  ( (token.length() ==  0)  | | (token.charAt(0) !=  '/') )
                                tag[count++] =  new  Tag(token,  true);  // opening tag
                            else  // ending tag
                                tag[count++] =  new  Tag(token.substring(1),  false);  // skip the
                    } // Note: we ignore anything not in an HTML tag
            }
    }
    return  tag;  // our array of tags
  }
  /** Tester method */
  public static void  main(String[ ]  args)  throws  IOException  {
                BufferedReader stdr;             // Standard Input Reader
                stdr  =  new  BufferedReader(new  InputStreamReader(System.in));
                HTML tagChecker  =  new  HTML();
    if  (tagChecker.isHTMLMatched(tagChecker.parseHTML(stdr)))
                System.out.println("The input file is a matched HTML document.");
                else
                System.out.println("The input file is not a matched HTML document.");
    }
}
```

# Implementation Example: Stack Interface in Java

- Java interface odgovara našem Stack ADT
- Zahteva definiciju klase EmptyStackException
- Razlikuje se od built-in Java klase java.util.Stack

```
public interface Stack {
    public int size();
    public boolean isEmpty();
    public Object top()
        throws EmptyStackException;
    public void push(Object o);
    public Object pop()
        throws EmptyStackException;
}
```

# Implementation Example (cont'd): Array-bazirani Stack u Javi

```
public class ArrayStack
      implements Stack {

   // holds the stack elements
   private Object S[ ];

   // index to top element
   private int top = -1;

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

```
   public Object pop()
         throws EmptyStackException {
      if isEmpty()
         throw new EmptyStackException
            ("Empty stack: cannot pop");
      Object temp = S[top];
      // facilitates garbage collection
      S[top] = null;
      top = top – 1;
      return temp;
   }
```

# Redovi ADT

- Red ADT skladišti proizvoljne objekte
- Umetanja i brisanja prate first-in first-out šemu
- Umetanja su na zadnjem delu reda, a uklanjanja su na početku reda
- Glavne operacije reda:
  - enqueue(object): umeće element na kraju (pozadi) reda
  - object dequeue(): uklanja i vraća element na početnoj strani reda

- Dodatne operacije:
  - object front(): daje element na prednjoj strani bez uklanjanja
  - integer size(): daje broj uskladištenih elemenata.
  - boolean isEmpty(): označava da li ne postoje uskladišteni elementi

Exceptions

- Pokušaj izvršenja dequeue ili front na praznom redu baca EmptyQueueException

# Red primer

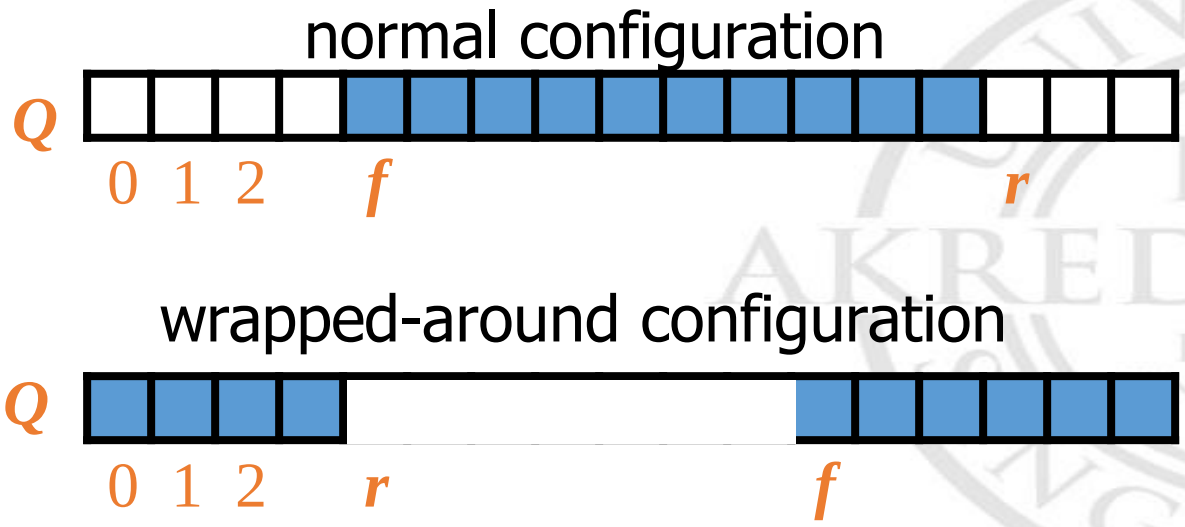
| *Operation* | | *Output* | *Q* |
|---|---|---|---|
| enqueue(5) | | – | (5) |
| enqueue(3) | | – | (5, 3) |
| dequeue() | *5* | (3) | |
| enqueue(7) | | – | (3, 7) |
| dequeue() | *3* | (7) | |
| front() | | *7* | (7) |
| dequeue() | *7* | () | |
| dequeue() | *"error"* | () | |
| isEmpty() | | *true* | () |
| enqueue(9) | | – | (9) |
| enqueue(7) | | – | (9, 7) |
| size() | | *2* | (9, 7) |
| enqueue(3) | | – | (9, 7, 3) |
| enqueue(5) | | – | (9, 7, 3, 5) |
| dequeue() | *9* | (7, 3, 5) | |

# Aplikacije redova

- Direktne aplikacije
  - Lista čekanja, javne usluge, banka, aerodromi
  - Pristup deljenim resursima(npr. printer)
  - Multiprogramming
  - Search algoritmi
- Indirektne aplikacije
  - Pomoćne strukture podataka za algoritme
  - Komponenta drugih struktura podataka

# Array-bazirani Redovi

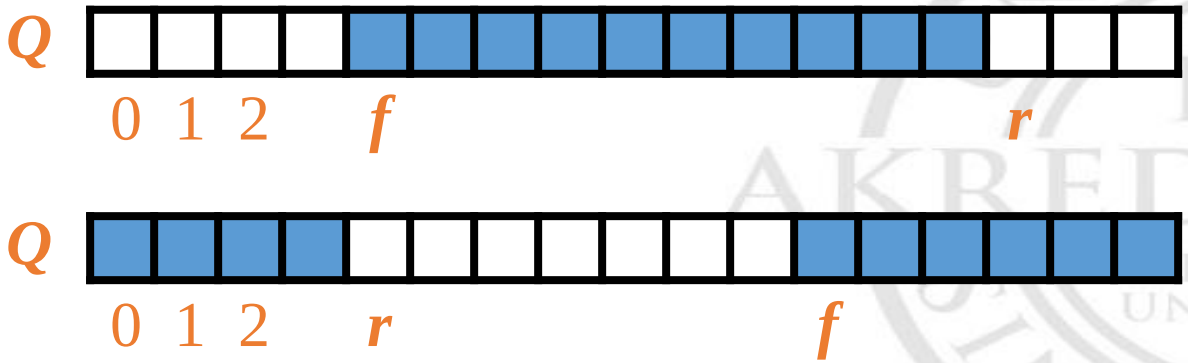
- Korišćenje niza veličine $N$ na kružni način
- Dve promenljive prate prednju i zadnju stranu
  - $f$ indeks prednjeg elementa
  - $r$ indeks odmah pored zadnjeg elementa
- Lokacija niza $r$ čuva se prazna
  - Red može da podrži $N-1$ elemenata



# Operacije redova (cont.)

- Operacija enqueue baca izuzetak ako je niz pun
- Ovaj exception je implementaciono zavistan

**Algorithm** *enqueue*(*o*)
  **if** *size*() = *N* – 1 **then**
    **throw** *FullQueueException*
  **else**
    *Q*[*r*] ← *o*
    *r* ← (*r* + 1) mod *N*

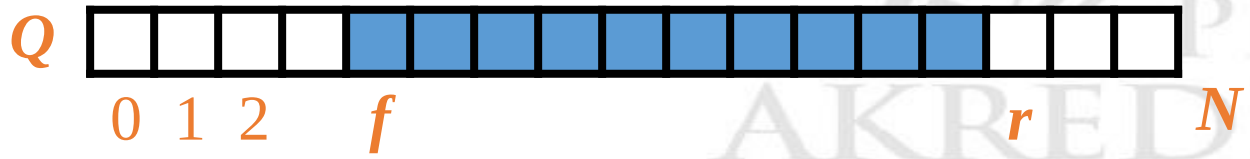


# Operacije redova (cont.)

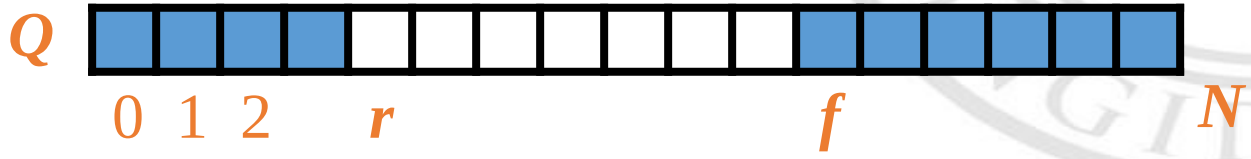
- Operacija dequeue baca izuzetak ako je red prazan
- Ovaj izuzetak je naveden u redu ADT

```
Algorithm dequeue()
  if isEmpty() then
    throw EmptyQueueException
  else
    o ← Q[f]
    f ← (f + 1) mod N
    return o
```

normal:




wrapped-around:



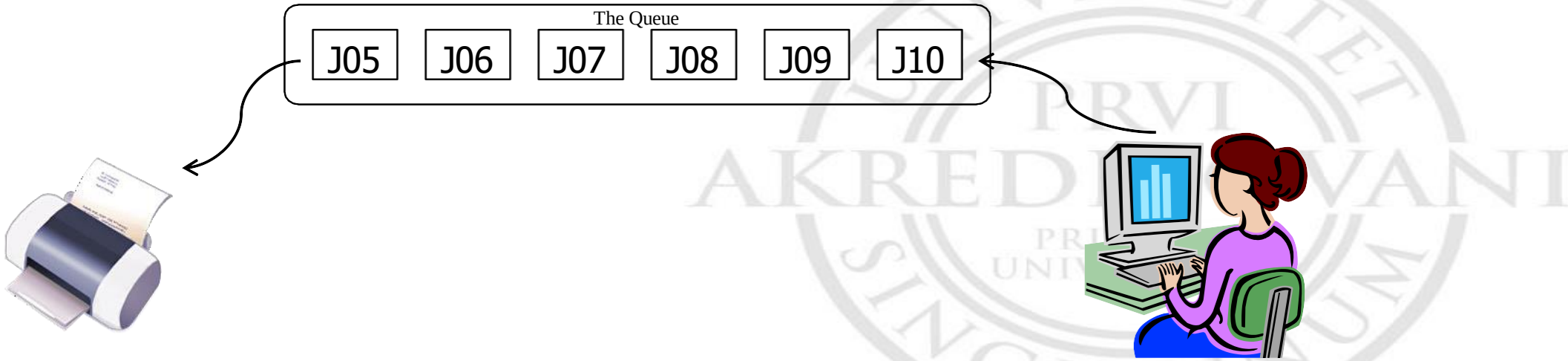
# Performanse i ograničenja

- Performanse
  - Neka $n$ bude broj elemenata u redu, i $N$ veličina niza
  - Prostor koji se koristi je $O(n)$
  - Svaka operacija se pokreće $O(1)$ vremena
- Ograničenja
  - Maksimalna veličina reda mora biti definisana apriori i ne može se menjati
  - Pokušavati da enqueue novi element u punom redu uzrokuje izuzetak specifičan za implementaciju
  - Kada je pun, potrebno je kreirati potpuno novi niz.
  - Ovaj problem se može rešiti pomoću niza koji se može proširiti.

# Aplikacije: Printer Queue

- Možemo da implementiramo printer job scheduler korišćenjem reda, *Q*, izvršavanjem sledećih koraka:

1. *e* = *Q*.enqueue() // send job u redosled štampača
2. *Q*.dequeue(*e*) // uzmi sledeći job iz reda
3. *e*.print() // poslati taj job na printer



# Primer implementacije: Queue Interface u Javi

- Java interface odgovara našem Queue ADT
- Zahteva definiciju klase EmptyQueueException
- Nema odgovarajuće built-in Java klase

```
public interface Queue {
   public int size();
   public boolean isEmpty();
   public Object front()
         throws EmptyQueueException;
   public void enqueue(Object o);
   public Object dequeue()
         throws EmptyQueueException;
}
```

# Reference

1. Algorithm Design and Applications by M. Goodrich and R. Tamassia, Wiley, 2015.
2. Data Structures and Algorithms in Java, 6th Edition, by M. Goodrich and R. Tamassia, Wiley, 2014.
3. Java Documentation (Java SE 8): http://www.oracle.com/technetwork/java/javase/overview/index.html
4. Automata Theory, Languages and Computation, 3rd Edition, by J. Hopcroft et al., Addison-Wesley, 2006.
5. Balanced parens solution without using a stack:

https://www.geeksforgeeks.org/check-for-balanced-parenthesis-without-using-stack/

# Review

1. Šta jeADT? Dajte primere.
2. Koje su glavne prednosti i mane array-based stacka? Isto tako za niz zasnovan na povezanoj listi.
3. Diskutujte o rešenju za slučaj u kojem niz postaje pun.
4. Dajte primer (svakog) ispravnog i netačnog podudaranja zagrada.
5. Dajte primere drugih HTML oznaka koje se mogu podudarati pomoću HTML algoritma podudaranja oznaka.
6. Dajte primer/problem u stvarnom životu koji se može primeniti/rešiti pomoću steka. Diskutujte o rešenju.
7. Šta je redADT? Dajte primere.
8. Koje su glavne prednosti i mane array-based reda? Isto tako za linked-list-based redove.
9. Implementiraj isEmpty operation. Koji su uslovi da red bude prazan?
10. Implementiraj the size() operaciju. Hint 1: možete koristiti if statement za dva slučaja (wrapped around or normal). Hint 2: koristite mod operator: (N + r – f ) mod N
11. Diskutujte o rešenju za slučaj u kojem jearray-based queue postaje pun.
12. Diskutujte oworst-case running time array-based enqueue and dequeue operacijama. Šta je sa najboljim slučajem?
13. Dajte primer/problem u stvarnom životu koji se može primeniti/rešiti pomoću reda. Razgovarajte o rešenju.