ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

One dimensional data structures

Algorithms and data structures

Exercise 5

One dimensional data structures

- Using the basic data structures new abstract data structures can be created
  - stack
  - queue
  - priority queue

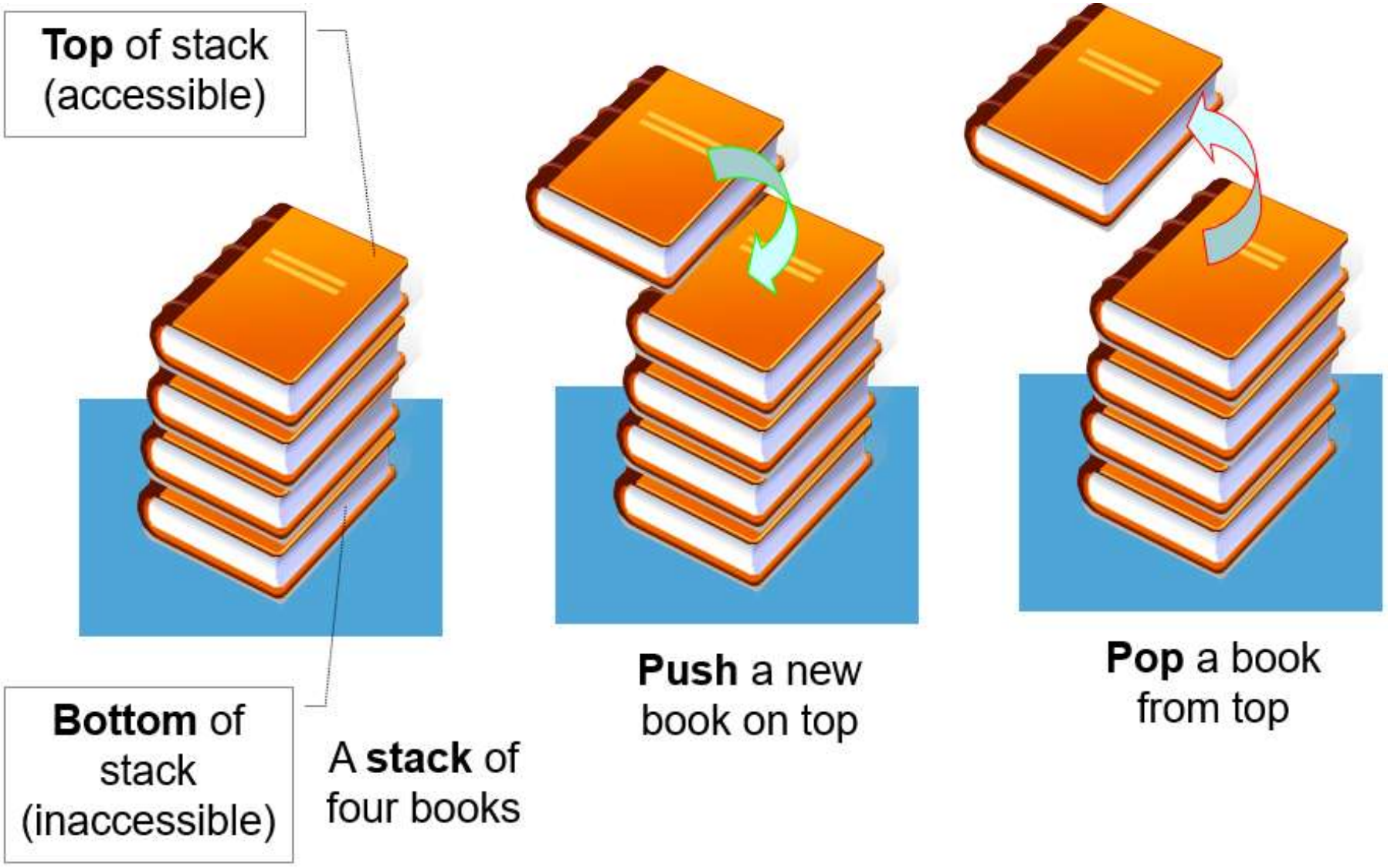
Stack

- **Stack** works with LIFO (Last In – First Out).
- Methods:
  - Adding an element at the top – push
  - Deleting an element from the top – pop
  - Additionally get the element from the top without delete - peek

Stack

- As one of the basic data structures used in many algorithms and applications
- Examples
  - Backtracking algorithms
  - Back/Forward browser functionalities
  - Undo/Redo functionalities
  - Reversing a list
  - Expression evaluation
  - Checking brackets in an expression
  - Parsers
  - Recursion
  - Function calls

Stack application



Stack implementation

- Two general stack implementations:
  - Limited stack – implemented with an array
  - Unlimited stack – implemented with a list

Stack interface - Java

```
public interface Stack<E> {
    // Elementi na stekot se objekti od proizvolen tip.
    // Metodi za pristap:

    public boolean isEmpty();
    // Vrakja true ako i samo ako stekot e prazen.

    public E peek();
    // Go vrakja elementot na vrvot od stekot.

    // Metodi za transformacija:
    public void clear();
    // Go prazni stekot.

    public void push(E x);
    // Go dodava x na vrvot na stekot.

    public E pop();
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
}
```

Used in both stack implementations

Stack with array - Java

```
public class ArrayStack<E> implements Stack<E> {
        private E[] elems; //elems[0...depth-1] se negovite elementi.
        private int depth; //depth e dlabochinata na stekot

        public ArrayStack(int maxDepth) {
            // Konstrukcija na nov, prazen stek.
        }

        public boolean isEmpty() {
            // Vrakja true ako i samo ako stekot e prazen.
        }
        public E peek() {
            // Go vrakja elementot na vrvot od stekot.
        }

        public void clear() {
            // Go prazni stekot.
        }

        public void push(E x) {
            // Go dodava x na vrvot na stekot.
        }

        public int size() {
            // Ja vrakja dolzinata na stack-ot.
        }
        public E pop() {
            // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
        }
    }
```

Stack with array - Java

```
public boolean isEmpty() {
    // Vrakja true ako i samo ako stekot e prazen.
    return (depth == 0);
}

public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (depth == 0)
        throw new NoSuchElementException();
    return elems[depth - 1];
}

public void clear() {
    // Go prazni stekot.
    for (int i = 0; i < depth; i++) elems[i] = null;
    depth = 0;
}
```

Stack with array - Java

```
public void push(E x) {
    // Go dodava x na vrvot na stekot.
    elems[depth++] = x;
}

public int size() {
    // Ja vrakja dolzinata na stack-ot.
    return depth;
}

public E pop() {
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
    if (depth == 0)
        throw new NoSuchElementException();
    E topmost = elems[--depth];
    elems[depth] = null;
    return topmost;
}
```

Stack with list - Java

```
public class LinkedStack<E> implements Stack<E> {
        // top e link do prviot jazol ednostrano-povrzanata
        // lista koja sodrzi gi elementite na stekot .
        private SLLNode<E> top;
        int size;

        public LinkedStack() {    }

        public boolean isEmpty() {}

        public E peek() {}

        public void clear() {}

        public void push(E x) {}

        public int size() {}

        public E pop() {}
    }
```

Stack with list - Java

```
public LinkedStack() {
    // Konstrukcija na nov, prazen stek.
    top = null;
    size = 0;
}

public boolean isEmpty() {
    // Vrakja true ako i samo ako stekot e prazen.
    return (top == null);
}

public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (top == null)
        throw new NoSuchElementException();
    return top.element;
}

public void clear() {
    // Go prazni stekot.
    top = null;
    size = 0;
}
```

Stack with list - Java

```
public E peek() {
    // Go vrakja elementot na vrvot od stekot.
    if (top == null)
        throw new NoSuchElementException();
    return top.element;
}

public void push(E x) {
    // Go dodava x na vrvot na stekot.
    top = new SLLNode<E>(x, top);
    size++;
}

public int size() {
    // Ja vrakja dolzinata na stekot.
    return size;
}

public E pop() {
    // Go otstranuva i vrakja elementot shto e na vrvot na stekot.
    if (top == null)
        throw new NoSuchElementException();
    E topElem = top.element;
    size--;
    top = top.succ;
    return topElem;
}
```

Class `Stack` in Java

- Class `Stack` is a LIFO object stack
- Inherited from class `Vector` with 5 new methods
- `public class` **`Stack<E>`** `extends` **`Vector<E>`**
- Methods:
  - `empty()`
  - `peek()`
  - `pop()`
  - `push(E item)`
  - `search(Object o)`

Problem 1.

- Check brackets correctness in an expression.
- An expression has correct brackets if:
  - For each left bracket, there is a right bracket
  - For each right bracket there is a previous left bracket
  - Each sub-expression between a pair of two brackets contains a correct number of brackets
- Examples:

$s \times (s - a) \times (s - b) \times (s - c)$ — correct

$(-b + \sqrt{[b^2 - 4ac]}) / 2a$ — correct

$s \times (s - a) \times (s - b \times (s - c)$ — not correct

$s \times (s - a) \times s - b) \times (s - c)$ — not correct

$(-b + \sqrt{[b^2 - 4ac)}] / 2a$ — not correct

Problem 1. - Java

```
public class Zagradi {
        public static boolean daliKorektni(String phrase) {
            // Test whether phrase is well-bracketed.
            ArrayStack<Character> bracketStack = new ArrayStack<Character>(100);
            for (int i = 0; i < phrase.length(); i++) {
                char cur = phrase.charAt(i);
                if (cur == '(' || cur == '[' || cur == '{')
                    bracketStack.push(cur);
                else if (cur == ')' || cur == ']' || cur == '}') {
                    if (bracketStack.isEmpty()) return false;
                    char left = bracketStack.pop();
                    if (!daliSoodvetni(left, cur)) return false;
                }
            }
            return (bracketStack.isEmpty());
        }
```

Problem 1. - Java

```
    public static boolean daliSoodvetni(char left, char right) {
        // Test whether left and right are matching brackets
        // (assuming that left is a left bracket and right is a right bracket).
        switch (left) {
            case '(':
                return (right == ')');
            case '[':
                return (right == ']');
            case '{':
                return (right == '}');
        }
        return false;
    }

    public static void main(String[] args) {
        String phrase = "s x (s - a) x (s - b) x (s - c)";
//        String phrase = "s x (s - a) x s - b) x (s - c)";
        System.out.println(phrase + " ima "
                + (daliZagraditeSePravilni(phrase) ? "korektni" : "nekorektni")
                + " zagradi.");
    }
}
```

Problem 2.

- HOMEWORK. Write an algorithm for postfix notation expression evaluation.
- Ex. 5 9 + 2 * 6 5 * + expression is in postfix notation

Queue

- **Queue** works with FIFO (First In – First Out).
- Methods:
  - Adding an element at the end – enqueue
  - Deleting an element from the beginning – dequeue
  - Additionally (in Java) getting the element fro the beginning (head) without deleting - peek

Queue - application

- With network rooters
- Playlists
- With network printers, where most users send documents for printing on a shared printer
- With hard disks, where most processes access disk data
- In operating systems, where most processes wait in line to be served from the processor

Queue implementation

- Two general queue implementations:
  - Limited queue – implemented with array
  - Unlimited queue – implemented with list

Queue interface - Java

Used in both queue implementations

```
public interface Queue<E> {
        // Elementi na redicata se objekti od proizvolen tip.
        // Metodi za pristap:
        public boolean isEmpty();
        // Vrakja true ako i samo ako redicata e prazena.

        public int size();
        // Ja vrakja dolzinata na redicata.

        public E peek();
        // Go vrakja elementot na vrvot t.e. pocetokot od redicata.

        // Metodi za transformacija:

        public void clear();
        // Ja prazni redicata.

        public void enqueue(E x);
        // Go dodava x na kraj od redicata.

        public E dequeue();
        // Go otstranuva i vrakja pochetniot element na redicata.
    }
```

Queue with array - Java

```
class ArrayQueue<E> {
        // Redicata e pretstavena na sledniot nacin:
        // length go sodrzi brojot na elementi.
        // Ako length > 0, togash elementite na redicata se zachuvani vo
    elems[front...rear-1]
        // Ako rear > front, togash vo  elems[front...maxlength-1] i elems[0...rear-1]
        E[] elems;
        int length, front, rear;

        // Konstruktor ...
        public ArrayQueue(int maxlength) {
            elems = (E[]) new Object[maxlength];
            clear();
        }

        public boolean isEmpty() {
            // Vrakja true ako i samo ako redicata e prazena.
            return (length == 0);
        }

        public int size() {
            // Ja vrakja dolzinata na redicata.
            return length;
        }
        public void clear() {
            // Ja prazni redicata.
            length = 0;
            front = rear = 0;  // arbitrary
        }
```

Queue with array - Java

```
public E peek() {
        // Go vrakja elementot na vrvot t.e. pocetokot od redicata.
        if (length > 0)
            return elems[front];
        else
            throw new NoSuchElementException();
    }

    public void enqueue(E x) {
        // Go dodava x na kraj od redicata.
        if (length == elems.length)
            throw new NoSuchElementException();
        elems[rear++] = x;
        if (rear == elems.length) rear = 0;
        length++;
    }

    public E dequeue() {
        // Go otstranuva i vrakja pochetniot element na redicata.
        if (length > 0) {
            E frontmost = elems[front];
            elems[front++] = null;
            if (front == elems.length) front = 0;
            length--;
            return frontmost;
        } else
            throw new NoSuchElementException();
    }
}
```

Queue with list - Java

```
public class LinkedQueue<E> implements Queue<E> {
       // Redicata e pretstavena na sledniot nacin:
        // length go sodrzi brojot na elementi.
        // Elementite se zachuvuvaat vo jazli dod SLL
        // front i rear se linkovi do prviot i posledniot jazel soodvetno.
        SLLNode<E> front, rear;
        int length;

        // Konstruktor ...
        public LinkedQueue () {
            clear();
        }

        public boolean isEmpty () {
               // Vrakja true ako i samo ako redicata e prazena.
            return (length == 0);
        }

        public int size () {
           // Ja vrakja dolzinata na redicata.
            return length;
        }
        public E peek () {
               // Go vrakja elementot na vrvot t.e. pocetokot od redicata.
            if (front == null)
                throw new NoSuchElementException();
            return front.element;
        }
```

Queue with list - Java

```
public void clear () {
        // Ja prazni redicata.
         front = rear = null;
         length = 0;
    }
    public void enqueue (E x) {
        // Go dodava x na kraj od redicata.
         SLLNode<E> latest = new SLLNode<E>(x, null);
         if (rear != null) {
             rear.succ = latest;
             rear = latest;
         } else
             front = rear = latest;
         length++;
    }

    public E dequeue () {
        // Go otstranuva i vrakja pochetniot element na redicata.
         if (front != null) {
             E frontmost = front.element;
             front = front.succ;
             if (front == null)  rear = null;
             length--;
             return frontmost;
         } else
             throw new NoSuchElementException();
    }
}
```

Queue interface in Java

- No queue class in Java, just an interface.
- `Queue` interface in Java:

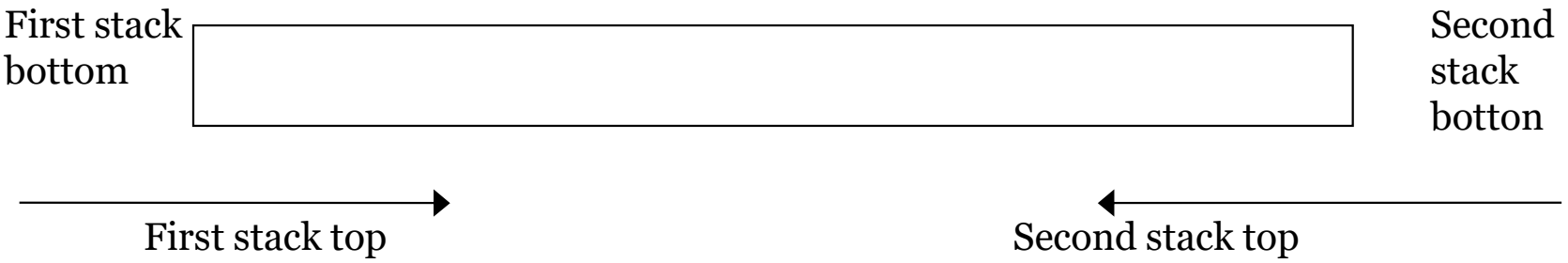
```
public interface Queue<E> extends Collection<E>
{
        E element();
        boolean offer(E e);
        E peek();
        E poll();
        E remove();
}
```

Queue in Java

- Classes that implement Queue interface are:
  - `AbstractCollection`
  - `LinkedList`
  - `PriorityQueue`
  - `LinkedBlockingQueue`
  - `BlockingQueue`
  - `ArrayBlockingQueue`
  - `LinkedBlockingQueue`
  - `PriorityBlockingQueue`

Problem 3.

- Write stack methods such that two stacks use shared space, as presented below.



Write functions to add and remove an element from the stacks, so that the stack used is determined by a variable passed as an argument to the corresponding functions.

Problem 3. - Java

```
public class DoubleArrayStack<E> {
        // Stekot e pretstaven na sledniot nacin:
        // depth1 e dlabochinata na prviot stekot,
        // depth2 e dlabochinata na prviot stekot,
        // elems[maxDepth-depth2...maxDepth-1] se elementi na vtoriot stek,
        private E[] elems; // elems[0...depth-1] se elementi na prviot stek,
        private int depth1;
        private int depth2;

        public DoubleArrayStack(int maxDepth) {
            // Konstrukcija na nov, prazen spodelen stek.
            elems = (E[]) new Object[maxDepth];
            depth1 = 0;
            depth2 = 0;
        }
        public boolean isEmptyFirst() {
            // Vrakja true ako i samo ako prviot stek e prazen.
            return (depth1 == 0);
        }
        public boolean isEmptySecond() {
            // Vrakja true ako i samo ako vtoriot stek e prazen.
            return (depth2 == 0);
        }
        public boolean isFull() {
            // Vrakja true ako i samo ako celata niza e polna.
            return (depth1 + depth2 == elems.length);
        }
```

Problem 3. - Java

```
public E peekFirst() {
        // Go vrakja elementot na vrvot od prviot stek.
        if (depth2 == 0)
            throw new NoSuchElementException();
        return elems[depth1 - 1];
    }

    public E peekSecond() {
        // Go vrakja elementot na vrvot od vtoriot stek.
        if (depth1 == 0)
            throw new NoSuchElementException();
        return elems[elems.length - depth2];
    }

    public void clearFirst() {
        // Go prazni prviot stek.
        for (int i = 0; i < depth1; i++)
            elems[i] = null;
        depth1 = 0;
    }


    public void clearSecond() {
        // Go prazni vtoriot stek.
        for (int i = elems.length - 1; i >= elems.length - depth2; i--)
            elems[i] = null;
        depth2 = 0;
    }
```

Problem 3. - Java

```
public void pushFirst(E x) {
        // Go dodava x na vrvot na prviot stek.
        if (!this.isFull())
            elems[depth1++] = x;
        else
            System.out.println("Error, the array is full");
    }

    public void pushSecond(E x) {
        // Go dodava x na vrvot na vtoriot stek.
        if (!this.isFull())
            elems[elems.length - (++depth2)] = x;
        else
            System.out.println("Error, the array is full");
    }


    public E popFirst() {
        // Go otstranuva i vrakja elementot shto e na vrvot na prviot stek.
        if (depth1 == 0)
            throw new NoSuchElementException();
        E topmost = elems[--depth1];
        elems[depth1] = null;
        return topmost;
    }
```

Problem 3. - Java

```
    public E popSecond() {
        // Go otstranuva i vrakja elementot shto e na vrvot na vtoriot stek.
        if (depth2 == 0)
            throw new NoSuchElementException();
        E topmost = elems[elems.length - depth2];
        elems[depth2--] = null;
        return topmost;
    }

    public String pecatiNizata() {
        StringBuilder ret = new StringBuilder("Elementite se: ");
        for (E elem : elems) ret.append(elem).append(" ");
        return ret.toString();
    }

    public static void main(String[] args) {
        DoubleArrayStack<Integer> d = new DoubleArrayStack<Integer>(6);
        d.pushFirst(1);
        d.pushFirst(2);
        d.pushFirst(3);
        d.pushSecond(-1);
        d.pushSecond(-2);
        d.pushSecond(-3);
        System.out.println("Vrv na prv: " + d.peekFirst() + ", dolzina na prv: " + d.depth1);
        System.out.println("Vrv na vtor: " + d.peekSecond() + ", dolzina na vtor: " + d.depth2);
        d.pushFirst(4);
        d.popFirst();
        d.pushFirst(4);
        System.out.println("Vrv na prv: " + d.peekFirst() + ", dolzina na prv: " + d.depth1);
        d.pecatiNizata();
    }
}
```

Priority queue

- One dimensional linear sequence, where each element has its own priority.
- The element with highest priority is always the first to be deleted
- The priority queue length is the number of elements it contains
  - Empty priority queue is with length 0

Priority queue

- Two possible implementations
  1. Ordered queue
  2. Unordered queue
- Both are independent of the data structure used (in performance)
  - With an array – limited priority queue
  - With a linked list – unlimited priority queue