

Лекция 1. OOP доц. Бирев

Структури данни

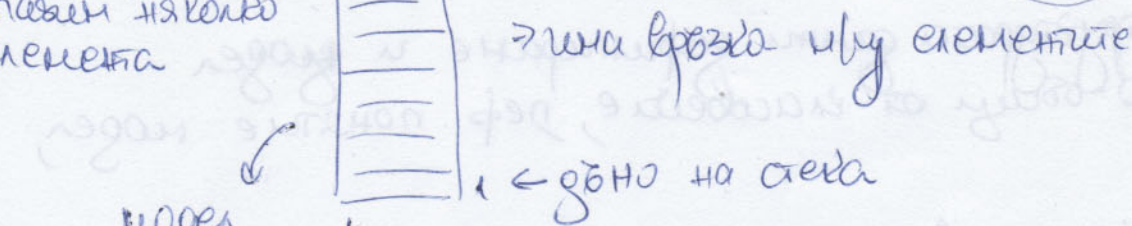
функция $\rightarrow$ (име на съдърж.)
данни

масив $\rightarrow$ описват начин, по който да работим с множество от
структури данни съвкупност множество елементи/обекти $\rightarrow$ единич. елементи

* class Complex $\rightarrow$ един или повече класа

class ArrayComplex $\rightarrow$ ина вътрешна архитектура

* Стек $\rightarrow$ множество
памят, в която
пазят няколко
елемента



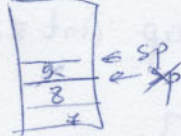
модел $\rightarrow$ начините на реализация а разликите
указател към върха на стека - stack pointer $\rightarrow$ посл. елемент
последен елемент $\rightarrow$ посл. елемент
който е записан
във върха на стека

отделение на интерфейса от дава реализацията

добавяне на елемент в стека push(x)
дава стойността на елемент из върха на стека top() $\rightarrow$ връща
изтрива елемент из върха на стека pop() $\rightarrow$ връща
резултат

връща резултат стойност на елемента с индекс == sp
имаме достъп до всеки един елемент, но не по всяко време

stack
stack.push(7);
stack.push(8);
stack.push(9);



инвариант на структура данни - ср. соем върха на стого-
вината (аксиома)

`x = stack.pop();`

`stack.pop();`

деф. строго опер. механизъм за
достъп до ел. на стека

алгебра на стр. данни (3-те операции - pop, top, push)
понятиеня система

структури данни - данни и операции с тях, групирани на едно
дуалност - данни - числови значения, програми се
функции състоят от тях

1 функция може да даде данни за друга
функция от по-висок ред - подава се функция като
масив от примитивна структура данни параметър

- идеята със структурата данни дефиниране и модел
стр. данни - по-общ от класовите, деф. понятие модел,
операции

- класовите се ползват за реализация на стр. данни

стр. данни - идеята е основна, тип
интерфейс и реализация, което съвпада и с обп
тип → интерфейс и клас

некоректни операции:

- празен стек

`boolean isEmpty()` → операция

`top()` - неопределена

stack Underflow

`pop()` - неопределена

- `push(x)` използване

stack Overflow

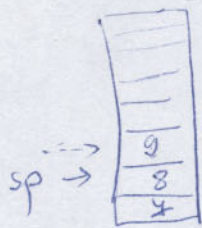
`class Stack`

`int[] st;`

`int sp;` може и с `new (int size)`

`public Stack() {`
`st = new int[100];`
`sp = 0;`
`}`

```
void push (int x) {
    sp = sp + 1;
    st[sp] = x;
}
```



// ани, ако стека е пълен
isFull()

if $sp == (st.length - 1)$ —
стека е пълен
игноматизирани изрази
if $sp == -1$ прасет стек

```
boolean isEmpty() {
    return (sp == -1);
}
```

```
boolean isFull()
{
    return (sp == (st.length - 1));
}
```

```
int top() {
    return st[sp];
}
```

```
void pop() {
    // isEmpty() != true (< sp[sp] = 0) → не-ефективно
    sp--;
}
// isEmpty() → if (!isEmpty())
```

```
if (!stack.isEmpty()) {
    stack.pop();
} ?
```

else
Syso ("Прасет стек");
Залучитно програмирање
надежност

main()

```
{
    // IntStack stack = new IntStack(1000);
    // ComplexStack cst = new ComplexStack(200);
}
```

готовност не зависи самостојно од типа
pop, top, push

```
class Stack <Type> {
    Type[] st;
    int sp;
```

Object

```
st = new Type[size];
      ↓
      не е важно !!
      ↑
push (Type x)
Type top()
```

```
main ()
```

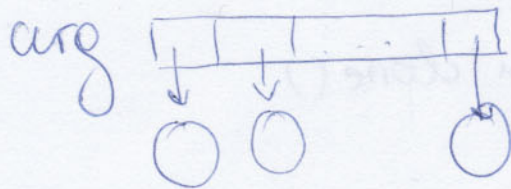
```
{
    Stack<Integer> st = new Stack(100);
    Stack<Complex> cst = new Stack(200);
```

но и това означава промяната
направена в класа е <Type>

Управление 1

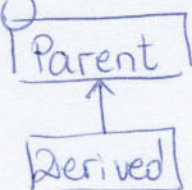
Задача 1

начало < край → инвариант на класа



$arg[i]$ → обект на класа отсечка

Задач 2



Constructor Parent(7)
Constructor Parent(-7)
Constructor Derived(7)

Constructor Parent() *този е погрешен*
Constructor Derived()

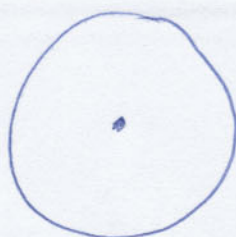
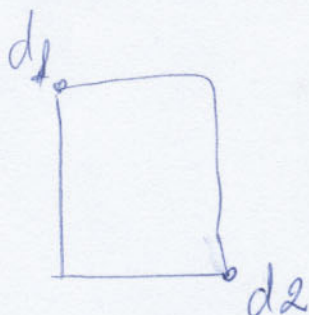
Example 2

Parent method 1
Parent method 2

Parent method 1
Derived method 1
Parent method 2

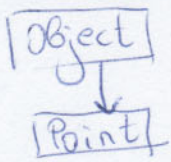
} 2

Задач



abstract - не могат да се създават екземпляри
от този клас

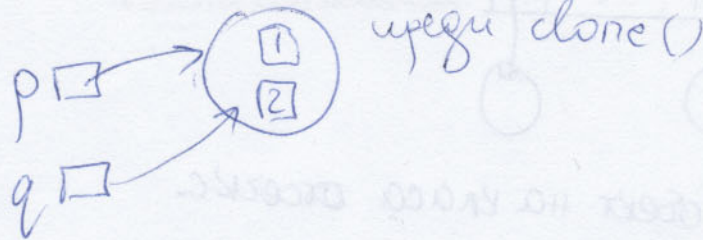
~~new Shape()~~



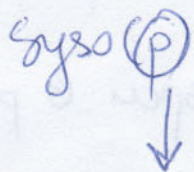
clone()
equals()
toString()

Point p = new Point(2, 3);

Point q = p;

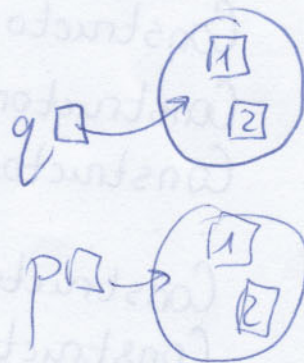


Point q = (Point) p.clone();



1 p.toString();

2 is being ce nonparent nes



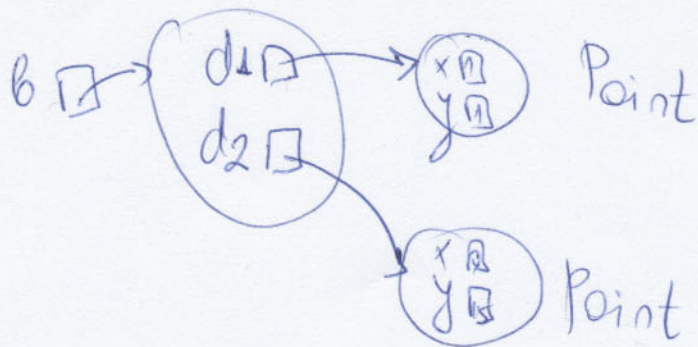
parent method 1
parent method 2
parent method 3
parent method 4



00n

Управление 2

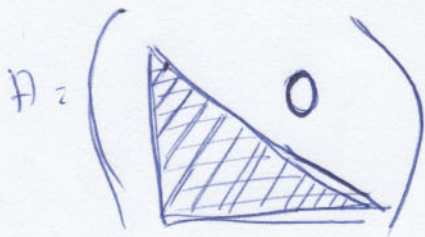
Block b = new Block (new Point(1,1), new Point(0,15))



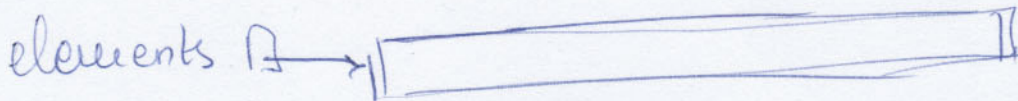
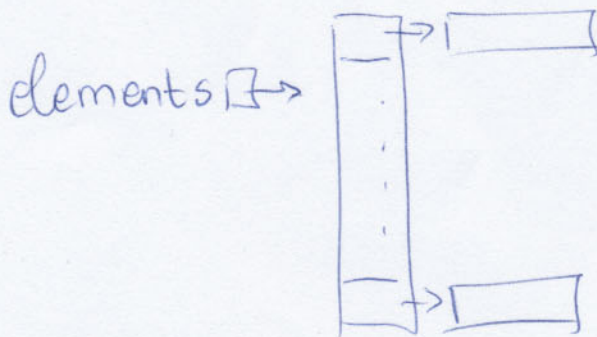
od_1

m

od_2



разредена матрица - 75% нули



Лекция 2 - COOP

- Литература:
- Cormen, Data Structure & Algorithms
 - Andrew Aho, John Hopcroft, Jeffrey Ullman
 - ✓ - Fundamentals of OOP & Data Structures in Java
Richard Wiener, Lewis Poonson Cambridge University Press
 - ✓ - Object-Oriented, Data Structures using Java
Nell Dale, Daniel Joyce, Chip Greems 2002
 - Data Structures and Algorithms with
OO Design Patterns in Java (C#, C++)
Bruno Preiss, Waterloo Canada, 2000

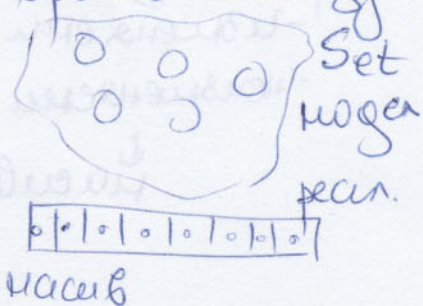
Гр. данни служат за съхраняване на елементи (едн./разн.)

Видове струк. данни:

- множествен (там елементите няма връзка помежду си
както еднотипни, разнотипни

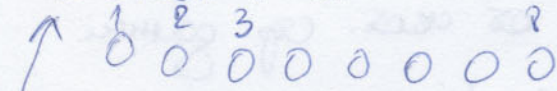
и ~~и~~ различни реализации
операции: 2 вида операции

- въвеждане → в/у елементите
- изваждане в/у цялата



обединение/сечение на две множества

- индексен тип

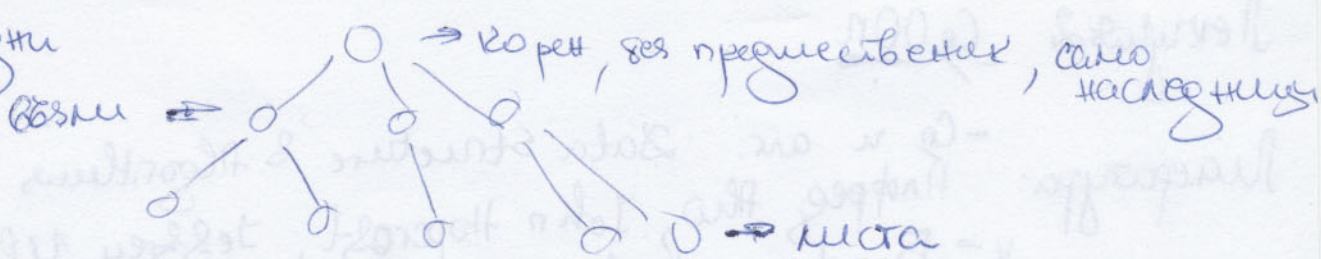


знае се за всеки елемент неговия предшественик и
следващият елемент

Списъчни структури

- * индексирани - гр. с пряк достъп
- * последователни - винаги започване от първия

- Дървовидни

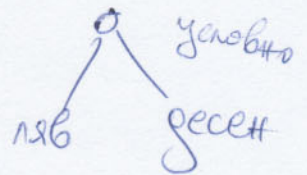


всеки елемент точно е един предшественик

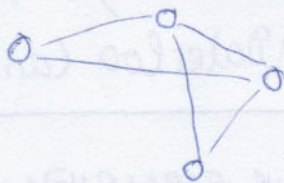
* типично дърв. структура - всеки елемент на следващото ниво

иерархически се структурират елементи

- мрежови (графоподобни)



* съседство, но няма иерархия

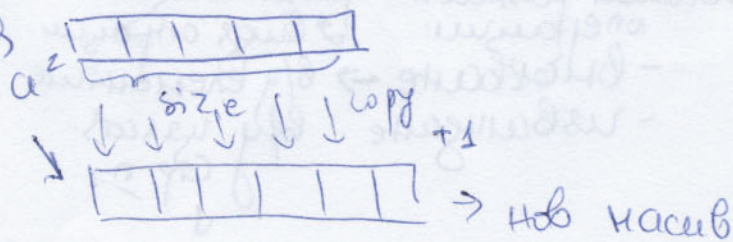


→ част от съседство в графа

според реализация:

- изменяеми → променя вер. архива, по време на изп. на прог.
- неизменяеми → не-и

масив



Видове операции:

мас. опер.

1. Конструктори - създават обекти от съот. ср. данни
2. Модификатори - променят съдърж./архива на ср. данни
 - добавяне и премаване на елементи
3. Инспектори - инспектират съдърж. на съот. ср. данни
 - Изм. isEmpty() дърв. харизм. за моментално състояние
 - search
4. Деструктори - разрушават ср. данни
 - clean() → премава всички ел. в стек
 - в начално положение
 - srz - 1

Лекция 3 C# 00n

Интерфейси

Що е то?

- спецификация на методи

- `interface Structure { extends списък-а - интерфейси }`

- {

- < специф. на методи >

- < декларация на полета >

- }

CS136 / Data Structures and Algorithms

Един клас може да реализира няколко интерфейса,
`CardInterface.SPADES`

1 и повече класа може да се дефинират в разн. интерфейси
Извикан с интерфейси

- Проектиране само с интерфейси

- `interface Set extends Structure { ... }`

- `interface Linear extends Structure { }`

- `- // - Tree - // -`

- `- // - Graph - // -`

Интерфейси и типове

- Тип? ост. операции

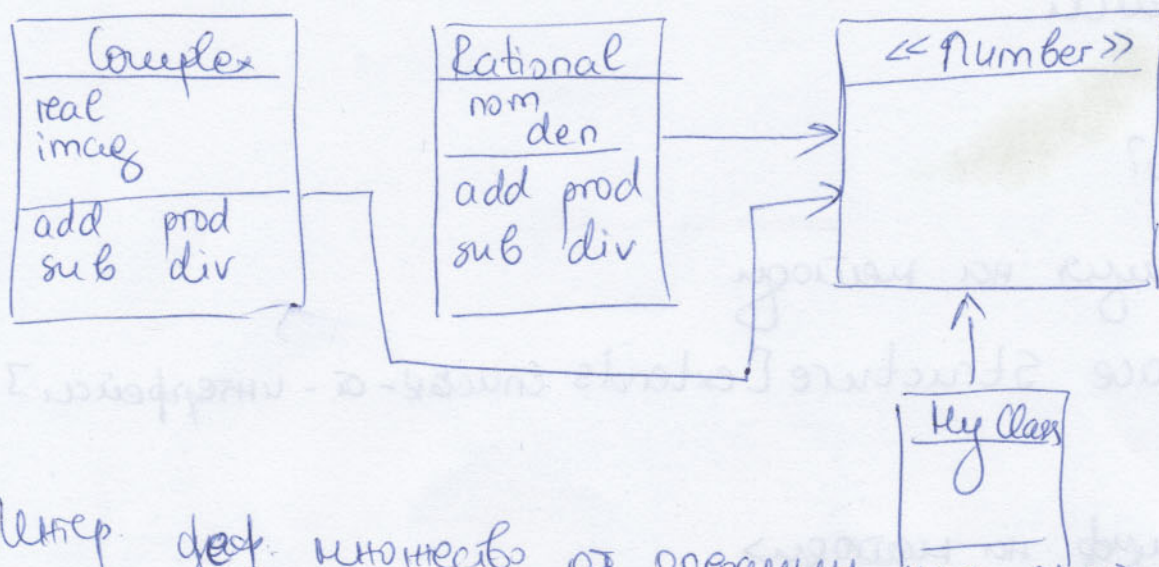
- Интерфейс

- тип

- Множество от методи

- Множество от стойности - от класа, който реализира този интерфейс

Нэколько класа рэалізуюць і і свае інтэрфейсы
- абшчност па інтэресах



Интер. д-р. множество от операции над своите свойства
 $\neq$ класове $\neq$ м-ва от свойства

разр. ин-вено от абилит. и ин-вено от операции
↳ конгрт. касе ↳ интерв

Резг. Типа на две части $\rightarrow$ интерфейс и клас

- Клас, който реализира интерфейс
 - всеки метод от интерфейса
 - какви и от кои методи без тях? (Не)
 - Клас - по-малък - възможна реализация на методите
 - Абстракция чрез спецификация
 interface → разширяване (клас)
 methods вече действат

- абстр.
- через спецификацию (отделяме важните неща от реал. → сигнатура)
 - чрез параметризация (типове или параметри)

Полиморфизъм

- По Полиморфно променлив
- В едно форми
- Приемат стойност от всеки клас, който реализира интерфейса

Number n = new Rational();

class Rational implements Number
n.add(Object o);

n = new Complex();

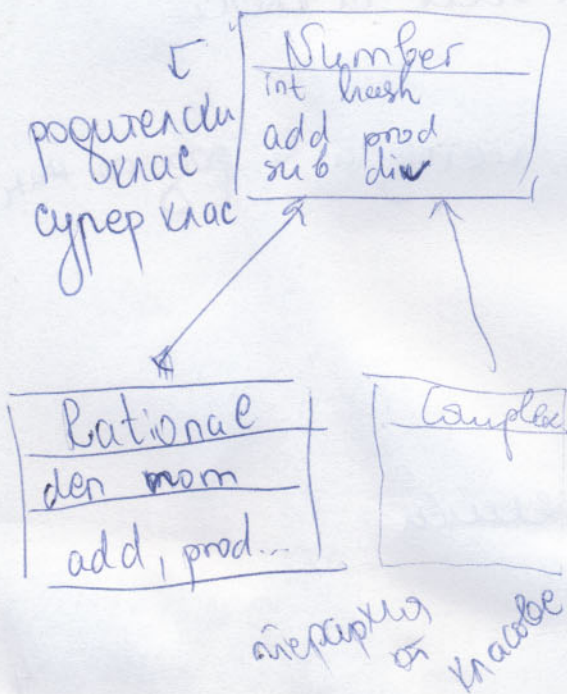
много форми

class Complex implements Number
n = new MyClass();

источник
полиморфност > интерфейс

Наследяване

P2 Inheritance and Refactoring



class Rational extends Number {

int num;
int den;
add(Object o)

→ произвежда
супер клас
потребителски клас

Complex c = new Complex();
 ↳ пометка на типа откъдето
 Rational

c.add(new Complex());

Rational

Number c = new Complex();

еднократно наследяване

множествено наследяване при интерфейсите (застава и няма само спецификация)

преизползване на код при подход клас

- позоваване на производни класове

- override

- this - дж. достъп към методите от подкласа

- super

private hash = 0;
 public Number() { ... }

public Rational() { ... }

hash = 0;
 super();
 this(...);

една и съща спецификация

→ някой констр. от Rational

от abstract class не може да се създават обекти

тази от класа е обект преизползване - mixing

обект от подклас може да замени обект от класа
 принцип на substituting

c.method() - в подтипа няма метод а в род то няма

↓
 (Rational) c.method()

↳ явно преобр. на тип -
 ↳ спецификация

Rational r = new Number(); → грешка

protected → ~~пр~~ ~~визит~~ се помещава в ~~гощер.~~ ~~класов~~
визит в ~~беларусия~~

class Rational extends Number implements ~~interface~~ NumberInterface {

Лекция 4 OOP

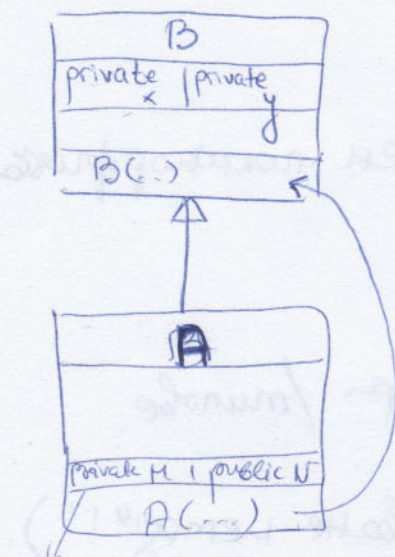
Наследяване

преизползване код

class A extends B → полета
→ методи

static - не се наследяват от B, общи
само полетата на екземпляра

private
protected
public



a = new A()

b = new B()

garbage collection - процедура за
сбориране на свободен памет

деструктори -

изящен номер на обекта.

инициал. само е разк. на B super()

```

A method()
{
    a = new A();
    return a;
}

B method()
{
    A b = method();
}
    
```

Преопределяне на методи
(overriding)

class B {

X method (min y) { ... }

}

class A extends B {

A < B

подклас

A a = new A();

X x = a.method(y);

class A extends B {
X method (mun x) {
System.out.println("Това е A.method!");
}
}

я аргумента пря от B е сменя с B

B b = new B();

X x = b.method(y);

b = new A();

X = ~~new~~ b.method(y);

overloading - претоварване (същият полиморфизъм)

X method (String str) {
Syso(str);
}

↓
различни параметри / типове

X = b.method("Това е претоварване метод!");

Абстрактни класове

[abstract] class <име>... {

abstract X method();

class B {...}

abstract class A extends B {

abstract void method A(...);

A a = ~~new~~ A();

```

class C extends A implements Int1, Int2 {
    // Int B = new Int1();
}
a = new C();    b = new C();

```

Вложенные классы

class A → приварено. соединены от функциональности

```

{
    min field 1; → область действия - ввв 1 класс
    class B { ... }
    class C { min2 field 2; }
    class D { ... }
}

```

→ минимальное требование от
 class C { min2 field 2; } → всего так независимы
 class D { ... } → field 1 на класс A

Защитно-программируемые

- аварийный режим, от не гурим данные

1 // (1-1); → неофициально

- писка на дисково пространство
- использование на ОЗ

стек - overflow

используя стек underflow

Object top()

```

{
    return st[sp];
}

```

if (sp == -1)
 return null;

```
Stack st = new Stack();
```

```
Object o = st.top();
```

```
if (o != null)
```

```
    ↳ !st.isEmpty();
```

```
    {
```

```
        o = st.top();
```

```
    }
```

Исключения вызываются

Исключения

Exception

→ от передачи от
языка на Java

↓
класс, с которым связан. user

MyException

вызывает

```
class StackOverflow extends Exception
```

```
{
```

```
    public StackOverflow (String msg)
```

```
    {
```

```
        super(msg);
```

```
    }
```

```
}
```

```
class StackUnderflow extends Exception
```

```
{
```

```
    public StackUnderflow (String msg)
```

```
    {
```

```
        super(msg);
```

```
    }
```

```
}
```

```
StackOverflow so = new StackOverflow("XXXXX  
user!");
```

StackOverflow вызывает
в методе class #1

Лекция 5 OOP

Защитно програмиране

Изключения

Декларация

Exception

1. Клас

2. променлива (анонимна)

Употреба:

`IndexOutOfBoundsException`

3. Обработка на изключения

4. Възбуждане на изключения

```
method A()  
{  
    a[-15] = a[-15] + 1;  
}
```

ако

→ ниста на аварийно изпълнение на програмата
изключението се разпространява към метода,
в който е оператора, ако не е в блок за
изключение

```
method B()  
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

```
{
```

`method B();` прекратява се методът B и, ако
той е извикан от метод C и той
се прекратява

отка и го метод main
`main()`

```
method C();  
}
```

излезе списък с грешки на решаване
true

Всички изключения са добре обработени
- програмата не трябва да завършва аварийно

5. Оператор за възбуждане на изключения
`throw <обект-изключение>;`

```
class StackOverflowException extends Exception {  
    public StackOverflowException(String msg) {  
        super(msg)  
    }  
}
```

```
StackOverflowException so = new StackOverflowException("Stack  
overflow!");  
throw so;
```

```
throw new StackOverflowException("Stack Overflow", index);
```

3. Обработчик на изключения

`<оператор-за-обработка-на-изключения> ::= <try-блок>
{ <обработчик-на-изключения> } [кlausal-finally]`

`<try-блок> ::=`

`try`
`{`

`<списък-от-оператори>`

`}`

`<обработчик-на-изключения> ::=`

`catch "(" декларация <тип-изключение> име на
променлива > ")"`

`{`

`<списък-от-оператори>`

`}`

```
catch (StackOverflowException ex)
```

```
{
```

```
    System.out.println(ex.getMessage());
```

```
    ex.printStackTrace();
```

```
}
```

! throws ex; e or more than one throw exception

```
catch (IndexOutOfRangeException ex)
```

```
{
```

```
    System.out.println("Index exception-2!");
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    System.out.println("Ya hoo");
```

```
}
```

} Here - throws

```
catch (StackOverflowException ex)
```

```
{
```

```
    System.out.println("Stack Overflow")
```

```
}
```

< finally - finally > :: =

finally → finally throws

"{"

< checked - or - unchecked >

"}"

```
if (ex is instanceof StackOverflowException)
```

```
{
```

```
}
```

Дизайн с исключениями

try { x=y; } catch (...) { ... }
try { y=y+1; } catch (...) { ... } → try {
 x=y; →
 y=y+1; (none goes to
 method A), continue B or
 finally)
} catch (...) { ... }

~~class~~ $\swarrow$
~~class~~ ... ~~throw~~
~~using~~ $[\text{throw n} < \text{structure} - \text{or} - \text{use} >]$

```
class Stack <T>
```

```
{ private T[] st;  
  private int sp;
```

```
public Stack (int size) {  
    st = (T[]) (new Object[size]);  
    (sp = -1).clear();  
}
```

```
void push(T item↑) throws StackOverflowException
```

```

    if (sp == (st.length - 1))
        throw new StackOverflowException("Stack
        Overflow");
}

```

(else)

$sp++$
 $st[sp] = item;$

3

Bruce Eckel

```
public void pop() T top() throws StackUnderflowException  
{  
    if (!isEmpty()) throw new StackUnderflowException  
        ("Stack Underflow!");  
    return st[sp]; // sp = sp - 1  
}
```

```
public boolean isEmpty()  
{  
    return (sp == -1);  
}
```

```
public void clear()  
{  
    sp = -1; // openpa  
            // for access null
```

void main()

```
{  
    Stack<Student> sts = new Stack<Student>(100);  
    try { sts.top(); } clear() → if (sts.isEmpty() == true) аканова  
        Student st = sts.top(); 38  
    } catch (StackUnderflowException ex) error  
    {  
        System.out.println("Не можем извлечь  
        элемент из стека!");  
    }
```

перемещение элемента на стек

Stack<Student> st2 = new Stack<Student>(1000);

```
while (!sts.isEmpty()) {  
    st2.push(sts.top());  
    sts.pop();  
}
```

создание

```

sts = st2;
sts.push(item);
}

```

```

return st2; // st2 is not a local variable
}

```

```

public boolean isEmpty() {
    return (st == null);
}

```

```

public void clear() {
    st = null;
}

```

```

void main() {
    Stack<Student> st = new Stack<Student>();
    Student st1 = new Student();
    st.add(st1);
    System.out.println("The number of students in the stack is: " + st.size());
}

```

```

Stack<Student> st2 = new Stack<Student>();
while (!st2.isEmpty()) {
    Student st3 = st2.pop();
}

```

Лекция 6 OOP

class Vector {

protected Object elementData[];

protected int elementCount; // текущее количество элементов

protected int capacityIncrement; //

protected final static int defaultCapacity = 10;

protected Object initialValue

зависит от метода в классе

v (v₁, v₂, ..., v_n)

АЛВ Вектор

изменяема

длина

- возможно добавление новых элементов

- 0 ... n-1 индекс элементов

↳

-1 < 0

↳ индекс элемента

public Vector() { this(defaultCapacity); }

public Vector(int initialCapacity)

{ // initialCapacity > 0

elementData = new Object[initialCapacity];

elementCount = 0;

capacityIncrement = 0;

initialValue = null;

}

public Vector(int initialCapacity, int capacityIncrement)

{ elementData = new Object[initialCapacity];

elementCount = 0;

this.capacityIncrement = capacityIncrement;

initialValue = null;

Vector v = new Vector();

↓
результат

конс. & се. элемент

начальное
значение

```
public Vector (int initialCapacity, int capacityIncrement,
               Object initialValue)...
```

```
public Vector (Vector that)
```

```
{
```

this ~~=~~ that ~~means~~! Is this the manner of copy constructor
 this (that.values());

```
}
```

```
public void add (Object item)
{
  1) if (elementCount == elementData.length)
     change
```

```
3) [element [elementCount] = item;
   elementCount++;
```

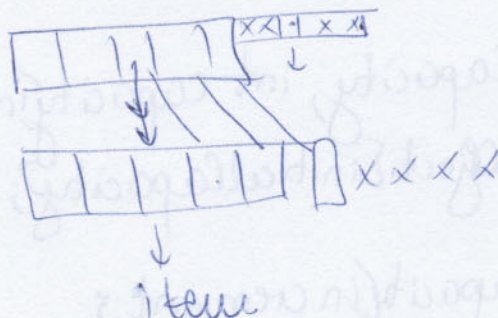
```
public Vector createVector (int initialCapacity)
```

1. upobepis 38 cb. nears
2. ybenuisbame elementData = capacityIncrement
3. gfoabame habus element



1. newElementData = new Object [elementData.length + capacityIncrement];
2. Compare us 4 or elem. new bl...
3. elementData = newElementData;

```
public void add (int index, Object item)
```



~~for~~ ^{usnors. or apus}
^{newage add}
 for (int i = elementCount, i > index, i--)
 elementData[i] = elementData[i - 1];
 elementData[index] = obj;
 elementCount++;

public void add(Object[] array) / void add (Vector v)
 private void

~~public~~ ensureCapacity (int ^{with} capacity) {
 if (elementData.length < minCapacity)
 {
 int newLength = elementData.length;
 if (capacityIncrement > 0)
 {
 if (newLength == 0) newLength = 1;
 while (newLength < minCapacity) {
 newLength *= 2;
 }
 }
 else {
 while (newLength < minCapacity)
 {
 newLength += capacityIncrement;
 }
 }
 }

→ Object newElement[] = new Object [newLength];
 for (int i = 0; i < elementData.length; i++)
 newElementData[i] = elementData[i];
 elementData = newElementData;

SK

3a of obodogum
 csp name

Deque

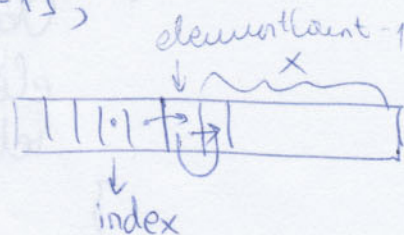
00n

```
public void add(Object item)
```

```
{
    ensureCapacity(elementCount + 1);
    elementData[elementCount] = item;
    elementCount++;
}
```

```
public void add(int index, Object item)
```

```
{
    ensureCapacity(elementCount + 1);
    for (int i = elementCount; i > index; i--) {
        elementData[i] = elementData[i - 1];
    }
    elementData[index] = item;
    elementCount++;
}
```



```
public void insertElementAt(Object item, int index)
{
    add(index, item);
}
```

```
public Object remove(int index) //
```

```
{
    Object res = elementData[index];
```

```
    elementCount--;
```

```
    while (index < elementCount)
```

```
    {
        elementData[index] = elementData[index + 1];
```

```
        index++;
```

```
    }
    elementData[elementCount] = null;
```

```
    return res;
```

```
}
```

Не може да се промени фоп. параметр при укажувањето
сув 6 success!

```
public Object Boolean removeElement(Object element)
```

```
{ int index = indexOfElement(element)
```

```
if (index == -1)
```

```
return false;
```

```
[Object x] remove(index);  
return true;
```

```
}
```

clear(); → премаха сите елементи на колекција

```
public Object set (int index, Object item)
```

```
{
```

```
Object previous = elementData[index];
```

```
elementData[index] = item;
```

```
return previous;
```

```
}
```

```
public void removeAllElements()
```

```
{
```

```
setSize(0);
```

```
}
```

```
public void setSize (int size)
```

```
{
```

```
if (size < elementCount)
```

```
{ for (int i = sizeelementCount; i < elementCount; i++)
```

```
elementData[i] = null;
```

```
}
```

```
else
```

```
// ensureCapacity(size);
```

```
for (int i = elementCount; i < size; i++)
```

```
elementData[i] = initialValue;
```

```
}
```

```
elementCount = size;
```

```
}
```

array[index] = item;

```

public int size()
{
    return elementCount;
}

```

```

main() elementCount = null
elementCount = new Object[10];
{
    Vector v = new Vector();
    v.add("abcd");
    v.add("1234");
    v.setSize(0); removeAllElements()
    v.add("Hloooooo!");
}

```

^{перес с нулевым индексом}
^{т.е. индекс}
^{т.е. индекс} ^{имеет значение}
^{наименование}

```

public boolean isEmpty()
{
    return (size() == 0);
}

```

```

public int indexOf(Object item) → индекс 18 е. 606
{
    буквы

```

```

public int lastIndexOf(Object item)

```

Съобщения

Лекция 8

```
public int indexOfElement(Object element)
{
    return indexOf(element, 0);
}
```

- /* предполагаем, че данните не са подредени
- * индексът намира индекса на елемент във вектор
 - * или връща -1, ако няма такъв елемент
 - * @param element стойност на елемента, който търсим
 - * @return индекс на елемент еквивалентен на element или -1, ако не е намерен
 - * @post връща индекса на елемента или -1, който записва търсене от началото. Вътрешният наредък не се променя
- */

→ пост условия
няма ги в Java

@pre → при какви положения няма да имаме изключителна ситуация.
елемент != null

indexOfElement(el, i) → @pre → index ga e в интервала 0...size()-1

design by contract

```
public int indexOfElement(Object element, int index)
{
    for (int i = index; i < elementCount; i++)
    {
        if (element.equals(elementData[i]))
            return i;
    }
    return -1;
}
```

boolean equals (Object o)

```
class Student [extends Object]
{
    int sacnom; String ime; int kus; int goup
```

boolean equals (Object o)

```
{
    Student that = (Student) o;
    return this.sacnom == that.sacnom;
}
```

не можно
е использовать
универсальный

↓
this.ime.equals(that.ime);
student.equals(new Complex());

Type Casting Exception

разные параметры - по-разному.

сравнение по методу → equals # Object
(пометки)

public int lastIndexOfElement (Object element, ^{int} index)

Object lastElement() { return getElement([↑]elementCount-1); }
Object firstElement() { return getElement(0); }
Object getNextElement (Object element)
{ return getElement(indexOfElement(element)); }

public Object clone() {
 Vector copy = null;
 try {
 copy = (Vector) super.clone();
 copy.elementData = (Object []) elementData.
 clone();
 }
}

```

catch ( java.lang.CloneNotSupportedException e)
{
    System.out.println("Vector cannot be cloned");
    return copy;
}

```

class Vector implements Cloneable

↓
clone();

```

public boolean contains (Element element)
{
    return (index of Element != -1);
}

```

```

public String toString()
{
    ↑ make it up
    StringBuffer sb = new StringBuffer();
    get indexes
    sb.append("[Vector:");
    for (int i=0; i<size(); i++)
    {
        sb.append(", " + get(i));
    }
    sb.append("]");
    return sb.toString();
}

```

```

void append (Vector v);
    ↓
    use this
    Vector append (Vector v);
    ↳ make it be better not better;

```

class VectorStack

usmen → usm.

```
{  
protected Vector st;  
protected Vector data;  
}
```

```
public VectorStack() { data = new Vector(); }
```

```
public void clear() { data.clear(); }
```

```
public void push(Object item) { data.add(item); }
```

```
public Object top() { return data.getlastfirstElement(); }
```

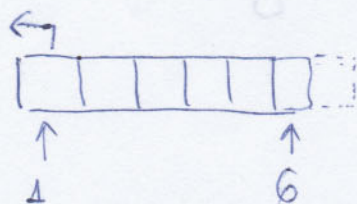
```
public Object pop()  
{  
    return data.remove(size() - 1);  
}
```

- toString -

Очередь

LIFO

FIFO



→ буфер

буфер на очередь

void enqueue(Object x) - добавляет элемент в очередь на очередь
Object dequeue() - извлекает элемент из очереди (премакивает)
Object front() - возвращает первый элемент из очереди

```
class ArrayQueue {
```

```
    private Object[] st;
```

```
    private int currentSize;
```

```
    private int front;
```

```
    private int back;
```

```
public ArrayQueue(int size) {
```

```
    st = new Object[size];
```

```
    front = 0;
```

```
    currentSize = 0;
```

```
    back = 0;
```

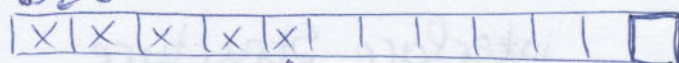
```
public void enqueue(Object item) {
```

```
    int back = (front + currentSize) % st.length;
```

```
    st[back] = item;
```

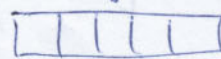
```
    currentSize++;
```

```
}
```

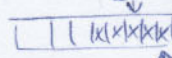


↑ back

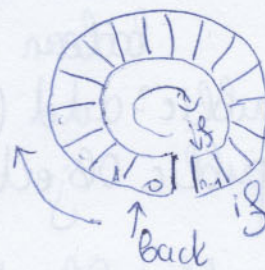
↓ back



↑ front



↑ back



↑ back

back = back + 1;

if (back == st.length - 1)

back = 0;

back = (back + 1) % st.length;

back = front - currentSize

интерпретация из class Queue

```
public Object dequeue()
```

```
{
    Object value = st[front];
    front = (front + 1) % st.length;
    currentSize--;
    return value;
}
```

деф. исключение

```
boolean isFull()
```

реализация через массив и буфер

```
{
```

- неизменяемая структура данных (через массив вектор)

```
boolean isEmpty()
```

- изменяемая структура

```
Object getFront()
```

зависит от реализации

```
interface Structure
```

```
{
```

```
// store
```

```
public int size();
```

// A₁ clear() → size() = 0

```
void clear();
```

```
boolean isEmpty();
```

```
public? add(Object value);
```

// A₂ size() = size() + 1

```
public Object remove();
```

- если не пуста, не можем
→ не вытаскиваем элемент

```
public Object remove(Object value)
```

```
boolean contains(Object value)
```

```
// SetStructure findAll(Object value)
```

```
Object[]
```

```
while (!isEmpty() || !contains(val))
```

```
{ Object val = r.h.a.g.remove();
```

```
str = str + val.toString();
```

```
}
```

```
return str;
```

копирование в StringBuilder

Сопремерни модели (design patterns)

Умератор

↳ построзване на ср. данни
↳ абстракция на поведението
↳ архитектура

java.lang.Enumeration

boolean hasMoreElements()

Object NextElement()

while (q.hasMoreElements())

str = str + q.NextElement();

java.lang.Enumeration values()

Enumeration iter ← q.values()

iter.hasMoreElements()

Лекция 10

CDu00n

java.util.Enumeration
java.util.Iterator

```
abstract class AbstractIterator implements Enumeration, Iterator
{
    public abstract Iterator() {}
    public abstract void reset();
    public abstract boolean hasNext();
    public abstract Object get();
    final public Object value() {
        return get();
    }
    public abstract Object next();
    final public boolean hasNextElements() {
        return next hasNext();
    }
    final public Object nextElement() {
        return next();
    }
}
```

```
class VectorIterator extends AbstractIterator
{
    protected Vector vector;
    protected int current;
    public VectorIterator(Vector v)
    {
        Vec vector = v;
        reset();
    }
}
```

```
public void reset() { current = 0; }  
public boolean hasNext()  
{ return (current < vector.size());
```

```
};  
public Object get()  
{ return vector.get(current);
```

```
};  
public Object next()  
{ return vector.get(current++);  
}
```

```
}
```

```
class Vector {
```

```
    public VectorIterator elements()  
    { return new VectorIterator(this);  
    }
```

```
};  
void main()
```

```
{ Vector v = new Vector();  
  v.add(...);  
  ...
```

```
  Iterator it = vectorv.elements();
```

```
  while (it.hasNext())
```

```
  { System.out.println(it.next().toString());  
  }
```

```
  it.reset();
```

```

public boolean contains (Object value)
{
    Iterator it = this.values();
    while ( it.hasNext() )
    {
        if ( it.next().equals(value) )
            return true;
    }
    return false;
}

```

```

interface Iterator < AnyType > extends java.util.Iterator < AnyType >
{
    boolean hasNext();
    AnyType next();
    void remove();
    void reset();
}

```

```

interface Linear extends Structure
{
    void add (Object value);
    public Object get();
    public Object remove();
    int size();
    boolean isEmpty();
}

```

```

abstract class AbstractLinear extends AbstractStructure
{
    public boolean empty()
    {
        return isEmpty();
    }
}

```

```

public Object remove()
{
    throw new Exception("Remove not implemented!");
    //return null;
}

```

```

}
abstract class AbstractStructure implements Structure
{
    public boolean isEmpty()
    {
        return (size() == 0);
    }
    public java.util.Enumeration elements()
    {
        return (AbstractIterator) iterator();
    }
    public boolean contains(Object value)
    {
        ...
    }
}

```

Примечание:

$$| \text{add}^n(x) = \text{add}(x) |$$

interface Set extends Structure

```

{
    public void addAll (Structure other)
    public boolean containsAll (Structure other)
    public void removeAll (Structure other)
    public void retainAll (Structure other)
}

```

→ this \ other → this
 → this & other ≤ this
 - equals

→ this n other → this
 не ∅ ≠ ∅

→ this \ other

abstract class AbstractSet extends AbstractStructure
implements Set

```
{  
    public void addAll (Structure other)  
    {  
        Iterator it = other.iterator();  
        while (it.next() hasNext())  
            this.add (it.next());  
    }  
}
```

- O(1) time complexity
- c Vector

Лекция 11
C2u00n

abstract class AbstractSet

extends AbstractStructure
implements Set

{

public void addAll(Structure other)

{

Iterator it = other.iterator();

while (it.hasNext())

add(it.next());

}

public void removeAll(Structure other)

{

Iterator it = other.iterator();

while (it.hasNext())

remove(it.next());

}

? public void retainAll(Structure other)

{

// неясно

}

class SetVector extends AbstractSet

{

protected Vector data;

public SetVector() {

data = new Vector();

← схема за
декларация

// декларатор

текстови файли

и др. в единичен

назив

// списък вектор,

```
public SetVector (Structure other)
{
    this();
    addAll (other);
}
```

```
public void clear()
{
    data.clear();
}
```

```
public boolean isEmpty()
{
    return
    data.isEmpty();
}
```

```
public void add (Object item)
{
    if (! data.contains (item))
        data.add (item);
}
```

```
public boolean containsAll (Structure other) // other  $\supseteq$  this
{
    Iterator elements = data.iterator();
    while (data elements.hasNext())
    {
        if (! other.contains (elements.next()))
            return false;
    }
    return true;
}
```

```
public void retainAll (Structure other)
{
    Set temp = new SetVector();
    Iterator it = data.iterator();
    while (it.hasNext())
    {
```

```

Object v = it.next();
if (other.contains(v))
    temp.add(v);

```

```

}
clear();
addAll(temp);

```

```

}

```

Списочные структуры данных (линейные, ассоциативные)

А. линейные списки

interface List extends Structure

```

{
    int size();
    boolean isEmpty();
    void clear();
    void addFirst(Object value);
    void addLast(Object value);
    Object getFirst();
    Object getLast();
    Object removeFirst();
    Object removeLast();
    void add(Object value);
    Object remove(Object value); // удалить элемент из списка
    Object remove(); // удалить элемент из списка
    Object get();
    boolean contains(Object value);
    public int indexOf(Object value);
    public int lastIndexOf(Object value);
}

```

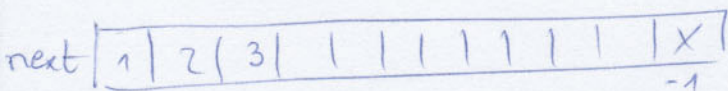
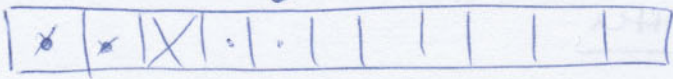
Object get (int value);
 Object set (int index, Object value);
 void add (int index, Object value);
 Object remove (int index);
 Iterator iterator();

28 апр

Контроль

список с вектор массив

1. Вектор реализует список

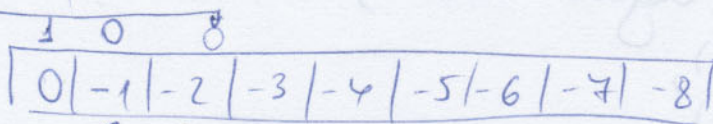
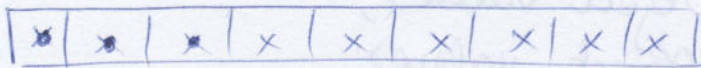


массив и

дополнительный массив с индексами

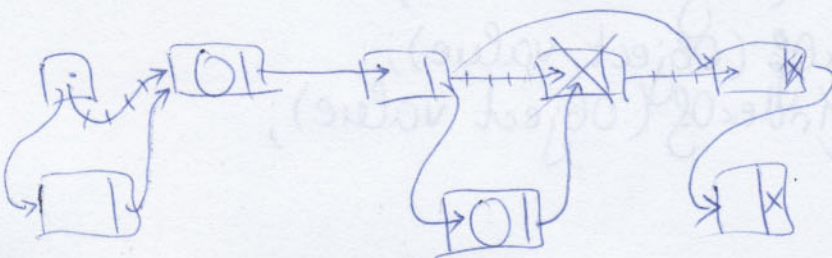
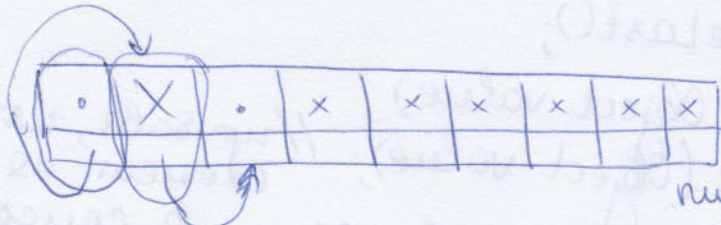
free ~~массив~~ ~~свободные~~ ~~элементы~~

↓ (с индексами - отрицательными)
 они все первые свободны



Free

Front



```

class SinglyLinkedListNode // listимеет односвязан список
{
    protected Object data; // рекурсивная структура данных
    protected SinglyLinkedListNode next;

    SLLNode(Object value, SLLNode next)
    {
        data = value;
        this.next = next;
    }

    SLLNode(Object value)
    {
        this(value, null);
    }

    public SLLNode next() { return this.next; }
    public void setNext(SLLNode next)
    {
        this.next = next;
    }

    public Object getValuevalue() { return datavalue; }
    public void setValue(Object value)
    {
        this.data = value;
    }

    public String toString() {
        return "[SinglyLinkedListNode(" + value() + ")]";
    }
}

```

```

class SinglyLinkedList extends AbstractList
{
    protected SinglyLinkedListNode head;
    protected int count;
}

```

SinglyLinkedList()

{

head = null;

count = 0;

}

посчитать и вернуть

```
class SLLNode {
    Object data;
    SLLNode next;
    SLLNode(data, next) {}
    SLLNode next() {}
}
```

```
class SList {
```

```
    SLLNode head;
    int count
```

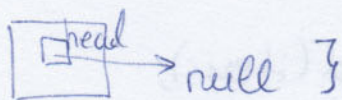
```
    void add(Object item)
    {
        //1.
```

```
        SLLNode temp = new SLLNode(item, null)
```

```
        //3 head = temp; count++;
```

```
        //2 temp.setNext(head);
```

```
        temp.next = head. (смена указателя)
```



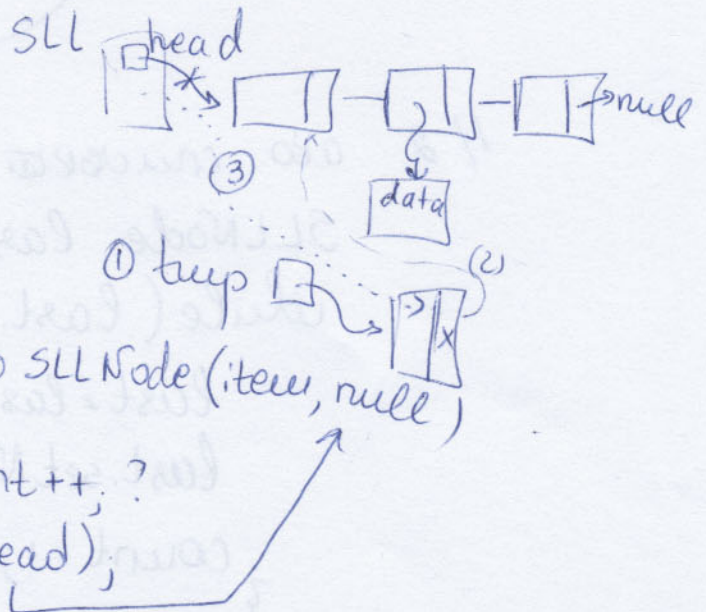
```
    head = temp
```

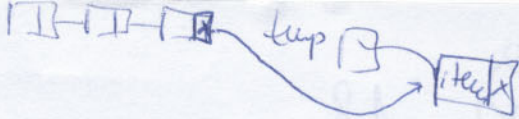
```
    addFront(Object item) {
```

```
        void add(Object item)
```

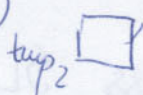
```
        {
            addFront(item);
```

```
        }
```



void AddRear(Object item) 

{ // 1. (*)

SLLNode tup = new SLLNode(item); 

head → 1-й элемент

head.next → 2-й элемент

head.next, ...

SLLNode tup2 = head;

tup2 = tup2.next

[tup2 = tup2.next];

while(tup2.next != null) {

// 2. also traverse the element

SLLNode last = head;

while (last.next != null) {

last = last.next();

last.setNext(tup);

count++;

}

// also traverse the element add(item);

(*) if (isEmpty()) {

add(item);

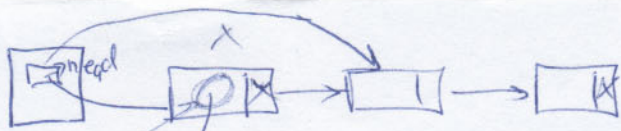
}

(...) + count(t₁ + t₂) +

O(n) + (t_{setNext}) + ...

(*) if (isEmpty()) head = tup; (можно добавить элемент)

как бы то и да е константа



val head = head.next()

// removeHead ^{работу выполняет, ако списъкът не е празен}
 Object removeHead() ^{if(isEmpty()) throw MyException}
 { // ↓ празен списък → return null;

// Object val = head.value(),

count--; ^{head = head.next();}
^{tmp.setNext(null);}
 return (val); ^{SLLNode tmp = head;}
 tmp.value();

main() { MyClass
 (Object) o = list.remove();
 MyClass

$O(1)$

MyClass m = list.remove();
 m.method(x); try/catch

NULLLIST - нощ. константа

↓
 не е празен, SLLNode
 когато м. е празен

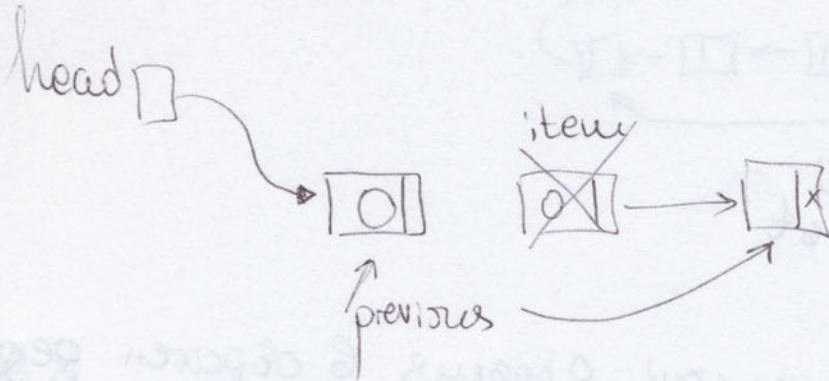
interface List

премахване итератор б/у

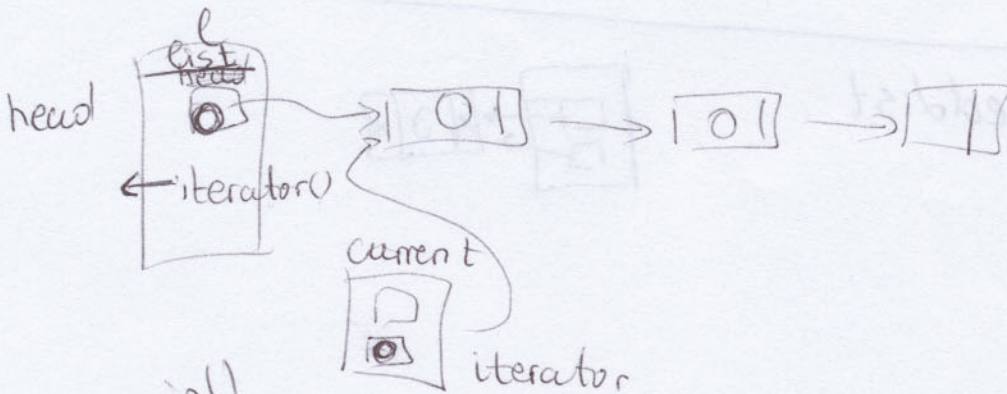
29.06 - nuchem usm
03.07 - yael usm

duane a. bailey

iff = if and only if

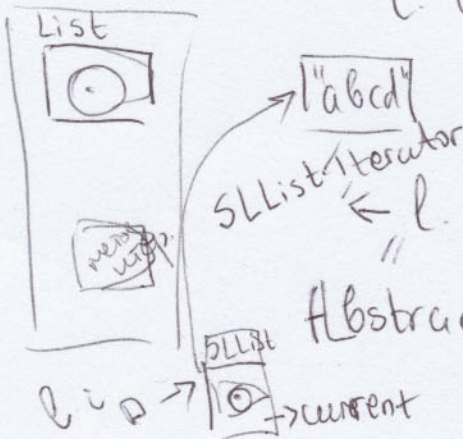


```
public Iterator iterator()
{
    return SinglyLinkedListIterator(head);
}
```

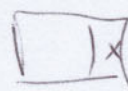


main()

```
List l = new SLList();
l.add("abcd");
l.add("xyzt");
```



AbstractIterator li =

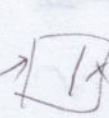
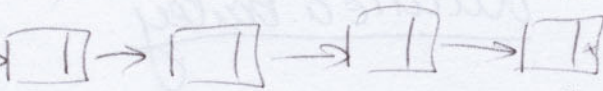


```
while (li.hasNext())
{
    doSomething(li.value());
    li.next();
}
```

for (AbstractIterator li = l.iterator(); li.hasMoreElements(); li.next())

$O(1)$ → добавление

previous $\rightarrow$ previous $\rightarrow$ $O(n)$



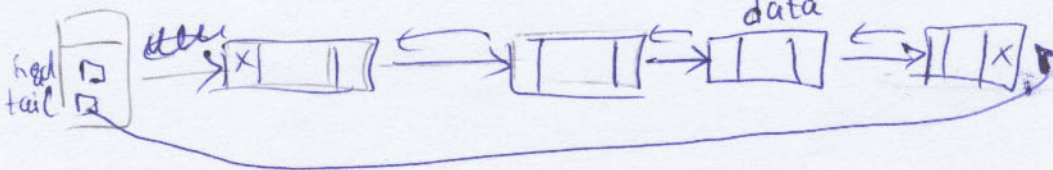
tail = previous
previous.setNext(next)



head = tail.next();
C SLList implements List

reverse() → список, который в обратном

$O(n)$



DoublyLinkedList



CDu00n
Лекция 14

class DataStructure implements DataStructureFIFO {

class Buffer {
 put(..)
 get(...)
}

Buffer bufLT450 = new Buffer();
Buffer bufGE450 = new Buffer();

class Iterator {

int from, to int current
Student[] data

Iterator (Student[] arr, int from, int to) {
 int remaining = arr.length - from; Buffer bf
 current = from; ✓
 Object next() {
 return data[current];
 remaining--;
 current++;
 }
}

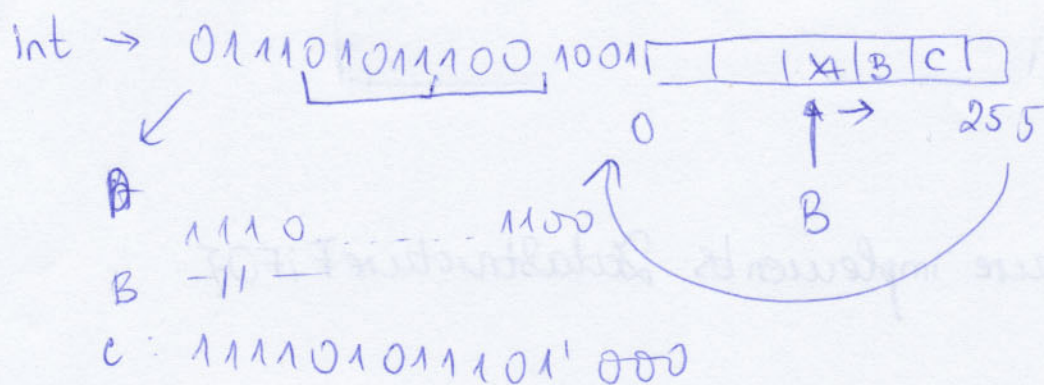
boolean hasNext() {

return ! (remaining == 0)

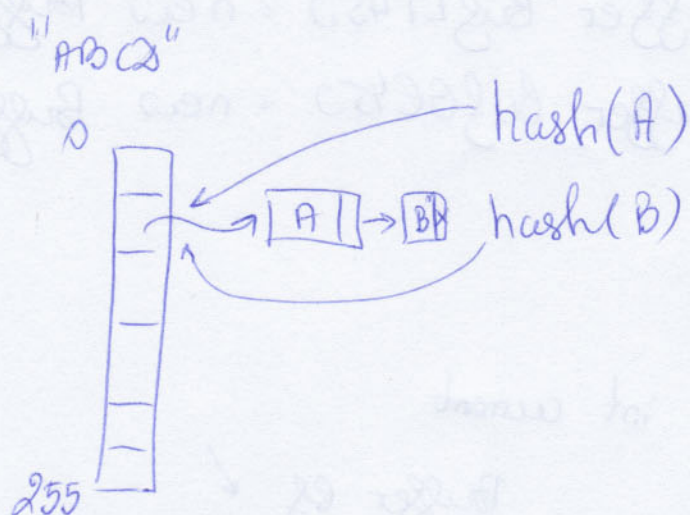
DataStructure (int size) {

bufLT450 = new BufferLT450 (size/2);

hash:



$A \% 256$

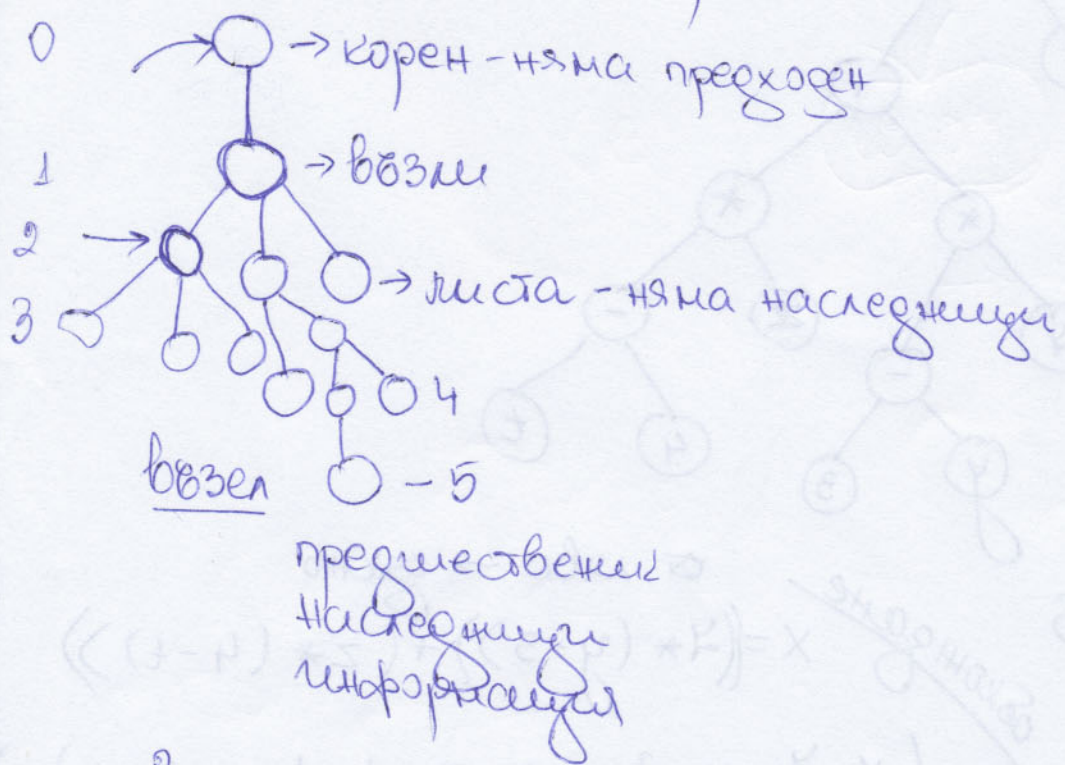


List[] association

A comparison structure - case of Bailey

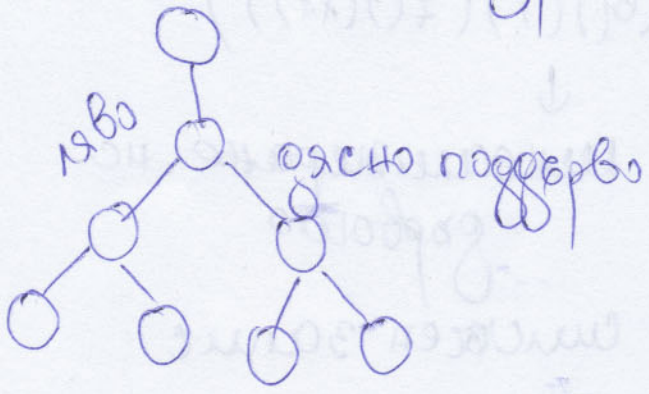
Лекция СДНООН

Иерархични структури данни Дървета.



2-и - бинарно
N-арно дърво } дефинираме дръз на наследниците

височина на възел - от възела до корена
височина на дърво - максимална височина на листо

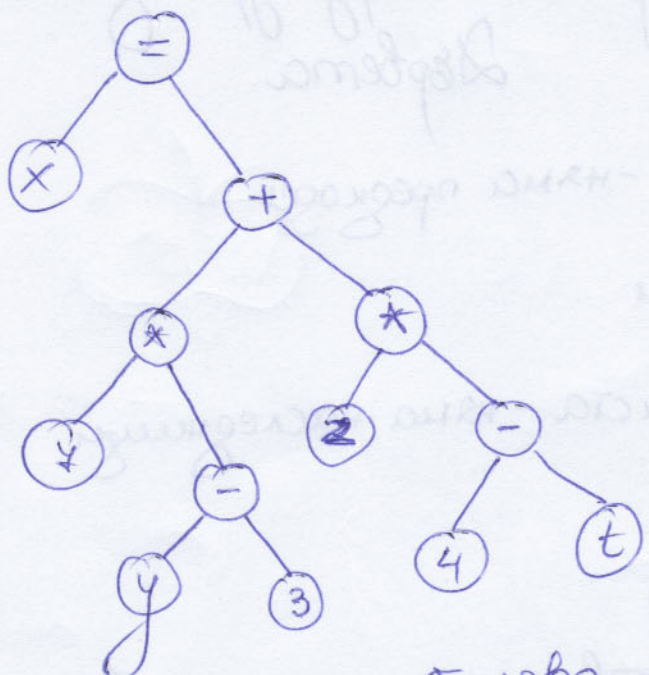


- синтактично дърво
чр. аритметичен израз

$$x = 4 * (y - 3) + 2 * (4 - t)$$

AST

Абстрактно
синтактично
дърво



while (B) S

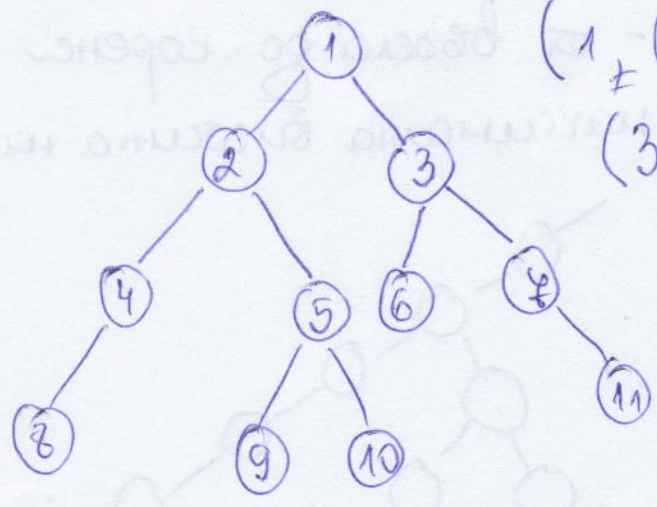
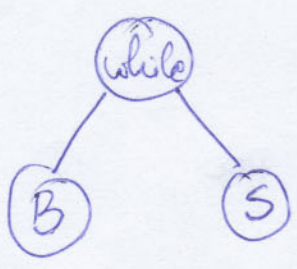
обхождане

от ляво → дясно

$$x = ((4 * (y - 3)) + (2 * (4 - t)))$$

(x, 4, y, 3, -, *, 2, 4, t, -, *, +) =

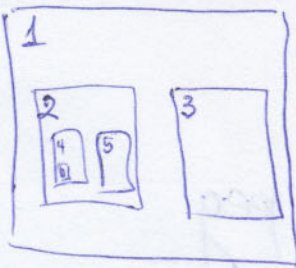
обратен полски запис
(стек)



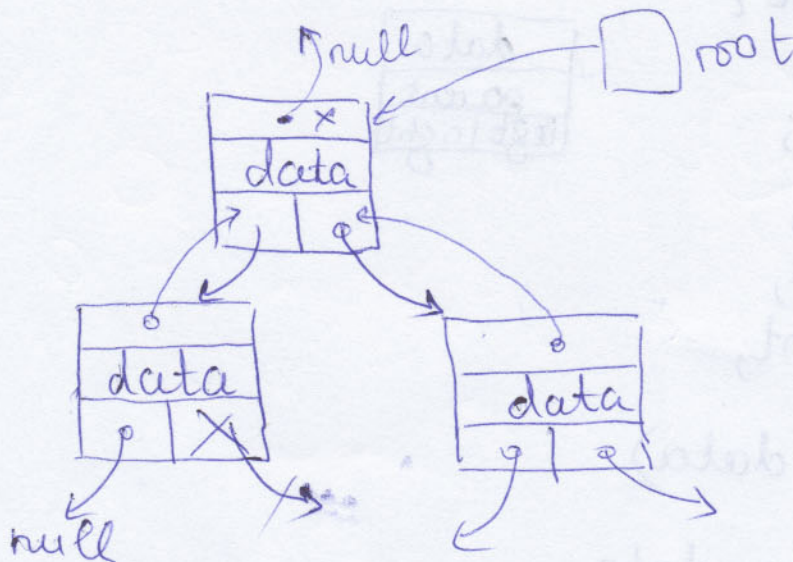
$(1 \pm (2(4(8)(1))5(9)(10)))$
 $(3(6(1)(1))(7(1)(11)))$

↓
интерпретиране на
дървото
символически запис

корен (ляво) (дясно)



ЕАН $9^{00}/10^{00}$
 Анализ - 210, 130, 8^{00}
 Геометрия 210, 9^{00}
 Queen - $9^{00}-20^{00}$
 $325/200$

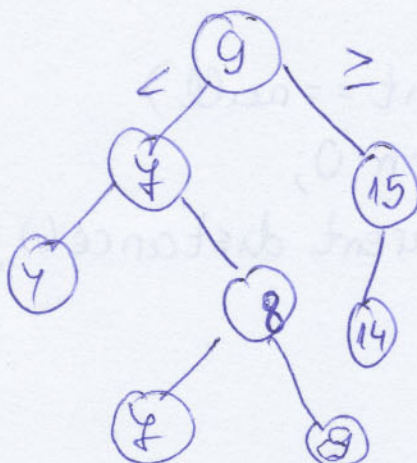


Звучи дървета

Звучи дървета за търсене

инварианти $<, =, >$

добавяне на сл. в др. дърво - за всеки възел всички ел. които се намират в лявото поддърво са по-малки от елемента, който стои във възела, съответно тези в дясното - по-големи



ляво поддърво, корен, дясно поддърво

4, 7, 7, 8, 8, 9, 14, 15, 33, → възходящ ред
дървовидна сортировка

```
class BinaryTreeNode {
```

```
    Object data;
```

```
    BinaryTreeNode left;
```

```
    BinaryTreeNode right;
```

```
    BinaryTreeNode parent;
```

```
    BinaryTreeNode (Object data)
```

```
    {
```

```
        this.data = data;
```

```
        parent = left = right = null;
```

```
    }
```

```
    BinaryTreeNode root() {
```

```
        if (parent == null)
```

```
            return this;
```

```
        return parent.root();
```

```
    }
```

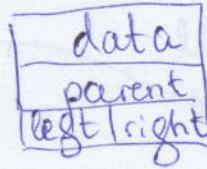
```
    int distance() {
```

```
        if (this.parent == null)
```

```
            return 0;
```

```
        return 1 + parent.distance();
```

```
    }
```



```
int depth() {
```

```
    if ((left == null) || (right == null))  
        return 0;
```

```
    return 1 + Math.max(left.depth(), right.depth());  
}
```

```
boolean isLeftChild() {  
    return parent.left == this;  
}
```

```
① if (left == null)
```

```
    {  
        if (right == null)
```

```
            return 0;
```

```
        else
```

```
            return 1 + right.depth();  
    }
```

```
    else if (right == null)
```

```
    {  
        return 1 + left.depth();  
    }
```

```
    }
```

```
    else
```

```
        return 1 + Math.max(left.depth(),  
                             right.depth());
```

```
    *
```

```
class BinaryTree {
```

```
    protected BTreeNode root;
```

```
    protected int count;
```

```
    * BTreeNode left()  
      right()  
      setLeft()  
      setRight()  
      < value()
```

```
interface Comparable {
```

```
    boolean lessThan(Comparable that);  
}
```

CDUON

values(Left) < root.value() ≤ values(Right)

add(Object value)
insert

interface Comparable {

int compareTo(Object value);

// result == 0 →

// < 0

// > 0

}

boolean lessThan (Object value)

add (Object value)

↑
Comparable

class Student implements Comparable {

int: compareTo (Object value) {

Student that = (Comparable) value;

return (this.sacnom < that.sacnom) ? -1;

return this.sacnom == that.sacnom ? 0, 1;

}

class BinarySearchTree implements Tree

{

void add (Comparable value) {

// this.isEmpty() ?

root = new BTR (value);

count++;

return;

```

if (value.compareTo(TreeNode tmp = root;root.value()) < 0) {
    while (true) {
        if (tmp.left == null) {
            tmp.setLeft(new BTreeNode(value, tmp, null, null));
            break;
        }
        tmp = tmp.left();
    }
} else {
    if (tmp.right == null) {
        tmp.setRight(new BTreeNode(value, tmp, null, null));
        break;
    }
    tmp = tmp.right();
}
}
}
current ++
}

```

boolean contains (Comparable value)

```

{
    BTreeNode tmp = root;
    if (root != null) {
        while (true) {
            if (tmp != null) {
                if (value.compareTo(tmp.value()) == 0) {
                    return true;
                }
                else if (value.compareTo(tmp.value()) < 0) {
                    tmp = tmp.left();
                }
                else {
                    tmp = tmp.right();
                }
            }
        }
        return false;
    }
}

```

```

class BTreeNode {
    BTreeNode parent
    static BTreeNode locate (BTreeNode root, Comparable value) {
        if (value.compareTo(root.value()) == 0)
            return root;
        if (root == null) return false;
        if (value.compareTo(root.value()) < 0)
            return contains (root.left, value);
        else
            return contains (root.right, value);
    }
}

```

```

boolean BST contains (Comparable value) {
}

```

```

    if (root == null) return false;
    BTreeNode temp = BTreeNode locate (root, root.parent(), value);
    if (value.equals (temp.value()))
        return true;

```

```

    else
        return false;
}

```

add

```

BTreeNode temp = BTreeNode locate (root, root.parent(), value);
if (value.compareTo (temp.value()) < 0)
    temp.setLeft (new BTreeNode (value, temp, null));
else
    temp.setRight (new BTreeNode (value, temp, null));
}

```

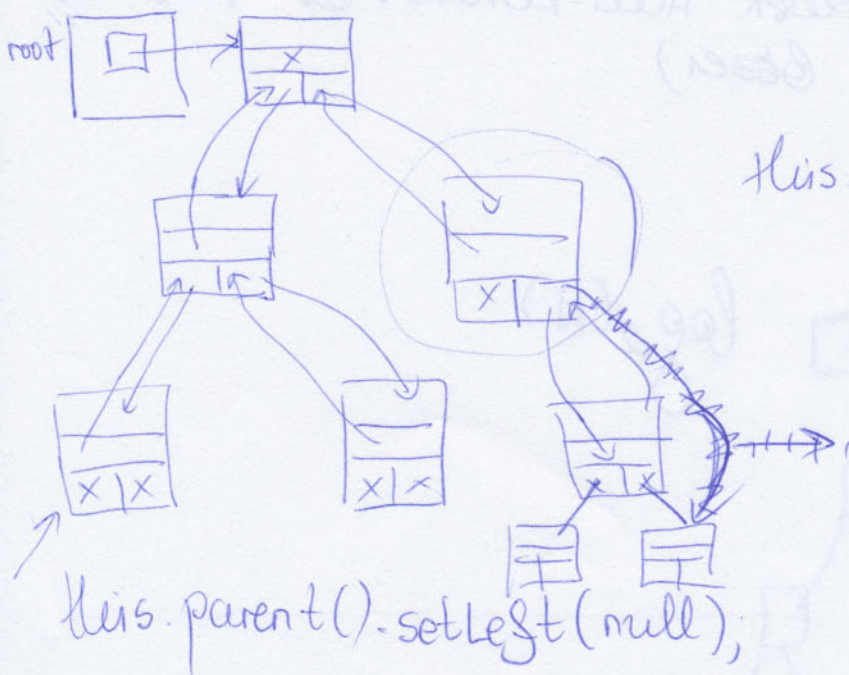
if (tup.right == null) tup.setRight(...)

BTNode tup1 = BTNode.locate(tup.right, tup, value)

if (tup1.setRight(new BTNode(value, tup1, null, null)))

3

С2u00n Лекция

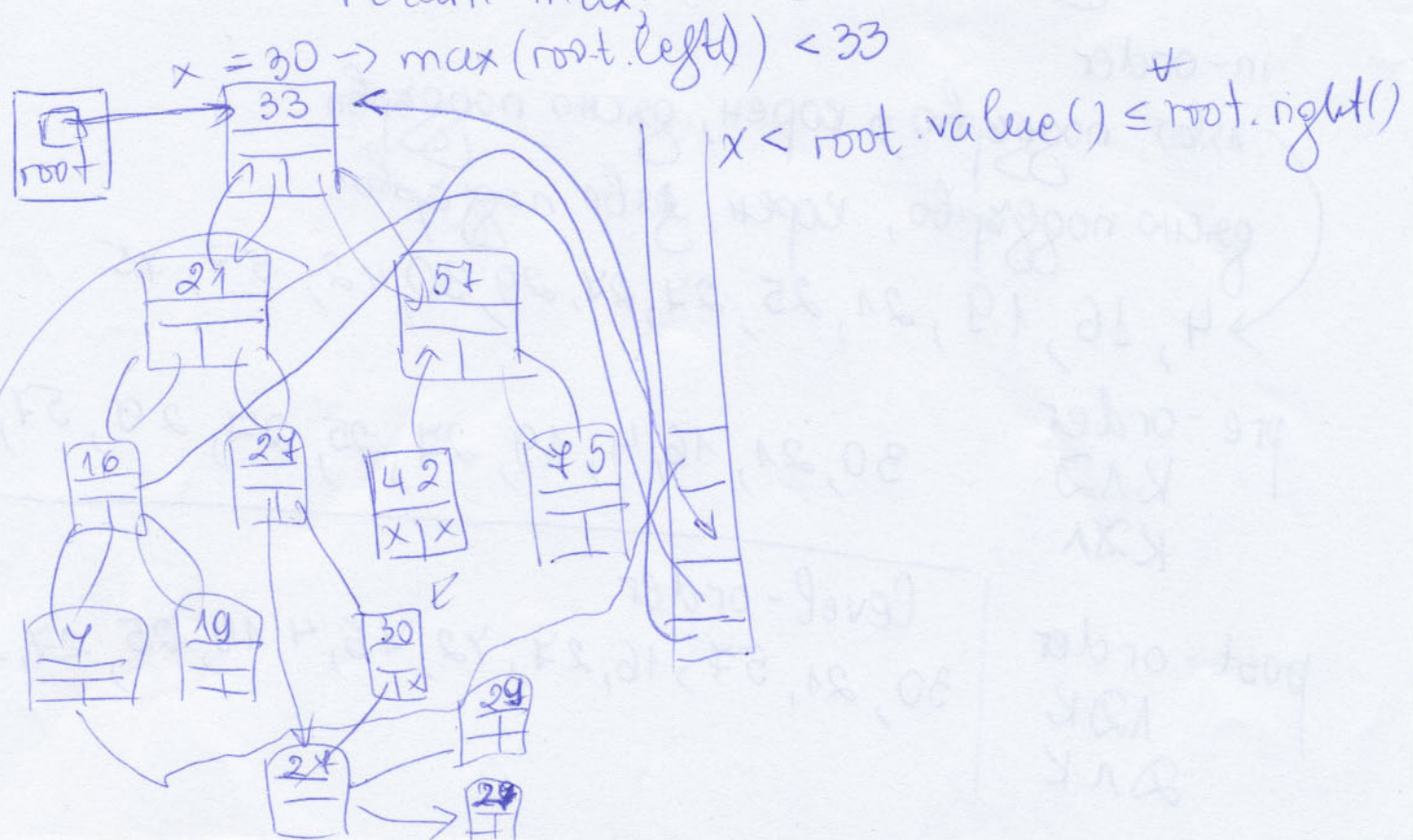


this.parent().setRight(this.right);

```

class BinaryTreeNode {
    public static BTreeNode max(BTreeNode root) {
        BTreeNode max = root;
        while (max.right != null)
            max = max.right();
        return max;
    }
}

```



30, (21, 57), 16, 24, 42, 45, 4, 19, 25, 27, null, null, null, null

корень $\rightarrow i$

лсбо $\rightarrow 2i+1$

дсбо $\rightarrow 2i+2$

$\rightarrow$ нулю зв. гербо в массив

parent $\rightarrow (i-1)/2$

boolean hasNext();

Object next();

Stack $\rightarrow$ Queue

СДшООН

Лекция

Графы

Множества структуры данных

$G(V, E)$

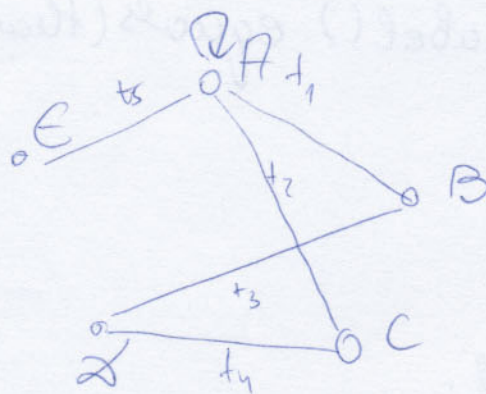
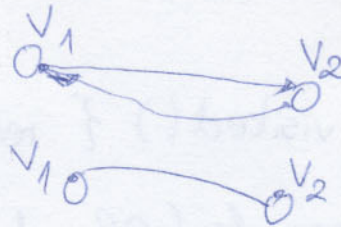
$V_1, V_2 \in V$

$e \in E$

$e \equiv (V_1, V_2)$

(V_2, V_1)

→ неориентирован



Таблица

	A	B	C	E	
A	1	1	1	0	1
B	1	1	0	1	0
C	1	0	1	1	0
E	0	1	1	1	0
	1	0	0	0	1

матрица из объектов

```
class Vertex {
```

```
    Object label;
```

```
    boolean visited;
```

```
    Vertex (Object label) {
```

```
        this.label = label;
```

```
        visited = false;
```

```
}
```

Object label() { return this.label; }

boolean visit()

boolean was = visited;

visited = true;

return was;

}

boolean is visited() { return visited; }

boolean equals(Object v)

{

Vertex that = (Vertex)v;

return this.label().equals(that.label);

}

class Edge {

Object v1 label;

Object v2 label;

Object label;

boolean directed;

boolean visited;

public Edge (Object v1, Object v2, Object label, boolean directed,

{

this.v1label = v1;

this.v2label = v2;

this.label = label;

this.directed = directed;

}

public Object here() { return v1label; }
public Object there() { return v2label; }
boolean visit() {

}

void setLabel(Object label) { this.label = label; }

Object label() { return this.label; }

boolean isDirected() { return directed; }

void reset() { visited = false; }

boolean equals(Object o) {

Edge e = (Edge) o;

return (here().equals(e.here()) &&

((there().equals(e.there())) && 11

{ ! directed &&

(here().equals(e.there()) &&

(there().equals(e.here()))));

}

interface Graph extends Structure {

public void add(Object label); // goes to 67x;

public void addEdge(Object v1, Object v2, Object label);

public Object remove(Object label);

public Object removeEdge(Object v1, Object v2);

public Object get(Object label);

```
public Object getEdge (Object v1, Object v2);  
Edge
```

```
boolean contains (Object label);
```

```
boolean containsEdge (Object v1, Object v2);
```

```
boolean void visit (Object label);
```

```
void visitEdge (Edge e);
```

```
void reset();
```

```
int size();
```

```
int edgeCount();
```

```
int degree (Object label);
```

```
public Iterator iterator();
```

```
public Iterator neighbours (Object label);
```

```
public Iterator edges();
```

```
void clear();
```

```
boolean isEmpty();
```

Not to read:

```
Graph g = reset;
```

```
g.reset;
```

```
if ( isPath (g, start, finish))
```

```
{ ...
```

```
}
```

```
boolean isPath ( Graph g, Object start, Object finish )  
{
```

```
    g.visit( start );
```

```
    if ( start.equals( finish ) )
```

```
        return true;
```

```
    else
```

```
    {
```

```
        boolean found = false;
```

```
        for ( Iterator = g.neighbours( start );
```

```
            it.hasMoreElements() && ! found;
```

```
            it.nextElement() )
```

```
        {
```

```
            Object neighbour = it.value();
```

```
            if ( ! neighbour.isVisited() )
```

```
                found = isPath( g, neighbour, finish );
```

```
        }
```

```
        return found;
```

```
    }
```

```
}
```

00n

:-)

Mercurus

class GraphMatrixVertex extends Vertex

```
{
    protected int index;
```

GraphMatrixVertex (Object label, int ind)

```
{
    super (label);
    index = ind;
}
```

int index () { return this.index; }

String toString () { ... }

~~abstract~~ } class GraphMatrix implements Graph

```
{
    protected int size;
    protected Edge [][] data;
    protected Map dict;
    protected List freeList;
    protected boolean directed;
```

GraphMatrix (int size, boolean directed)

```
{
    this.size = size;
    data = new Edge [size] [size];
    dict = new Hashtable (size);
    freeList = new SinglyLinked List ();
    for (int row = size - 1, row >= 0, row --)
        freeList.add (new Integer (row));
    directed = dir;
}
```

```

public void add(Object label)
{
    // no copy because it is a table element
    int row = ((Integer) freeList.removeFirst()).
        intValue();
    dict.put(label, new GraphMatrixVertex(label, row));
}

```

```

abstract void addEdge(Object vtx1, Object vtx2,
    Object label);

```

```

public Object remove(Object label) {
    GraphMatrixVertex vert = dict.remove(label);
    (GraphMatrixVertex)
    if (vert == null) return null;
    int index = vert.index();
    for (int row = 0; row < size; row++) {
        data[index][row] = null;
        data data[row][index] = null;
    }
    freeList.add(new Integer(index));
    return vert.label();
}

```

```

abstract Object removeEdge(Object vtx1, Object vtx2);
public Edge getEdge(Object v1, Object v2)
{
    int ind1 = ((GraphMatrixVertex) dict.get(v1)).
        index();
    int ind2 = ((GraphMatrixVertex) dict.get(v2)).
        index();
}

```

```

    return data[ind1][ind2];
}

public boolean containsEdge(Object v1, Object v2) {...}

public boolean visit(Object label)
{
    GraphMatrixVertex vert = (GraphMatrixVertex)
        dict.get(label);
    return vert.visit();
}

public boolean visitEdge(Edge e)
{
    return e.visit();
}

public int degree(Object label)
{
    int row = ((GraphMatrixVertex)dict.get(label)).
        index();

    int result = 0;
    for(int col = 0; col < size, col++) {
        if(data[row][col] != null) result++;
    }
    return result;
}

public Iterator iterator()
{
    return dict.keySet().iterator();
}

```

```
public Iterator neighbours (Object label)
```

```
{
```

```
    GraphMatrixVertex vert = (GraphMatrixVertex) dict.  
        get (label);
```

```
    int row = vert.index();  
    List list = new SLL();  
    for (int col = 0; col < size; col++)
```

```
    {
```

```
        Edge e = data [row] [col];
```

```
        if (e != null)
```

```
        {  
            if (e.vert.here().equals (label))  
                list.add (e.there());
```

```
            else if (e.there().equals (vert.label()))  
                list.add (e.here());
```

```
        }
```

```
    }
```

```
    return list.iterator();
```

```
}
```

```
abstract Iterator edges();
```

```
class UndirectedGraphMatrix extends GraphMatrix
```

```
{
```

```
    MGM (int size) { super (size, false); }
```

```
    public void addEdge (Object v1, Object v2, Object label)
```

```
    {
```

```
        GraphMatrixVertex vtx1 = (GMV) dict. get (v1);
```

```
        GMV vertex vtx2 = (GMV) dict. get (v2);
```

```
        Edge e = new Edge (vtx1, vtx2, label, false);
```

```
        data [vtx1.index()] [vtx2.index()] = e;
```

(*)
/ses here are
directed

```
(*) data [ vtx2. index() ] [ vtx1. index() ] = e;
}
```

```
Object removeEdge (Object v1, Object v2)
{
```

```
    int row = ... index of v1;
```

```
    int col = ... index of v2;
```

```
    Edge e = data [ row ] [ col ];
```

```
    data [ row ] [ col ] = null;
```

```
    data [ col ] [ row ] = null;
```

```
    if (e == null) return null;
```

```
    return e.label();
```

```
int edgeCount() { }
```

↳ спост на границе
Кд. - нечет

```
public Iterator edges()
{
```

```
    List list = new SLL();
```

```
    for (int row = size-1; row >= 0; row--)
```

```
        for (int col = size-1; col > row; col--)
```

```
        {
```

```
            Edge e = data [ row ] [ col ];
```

```
            if (e != null)
```

```
                list.add(e);
```

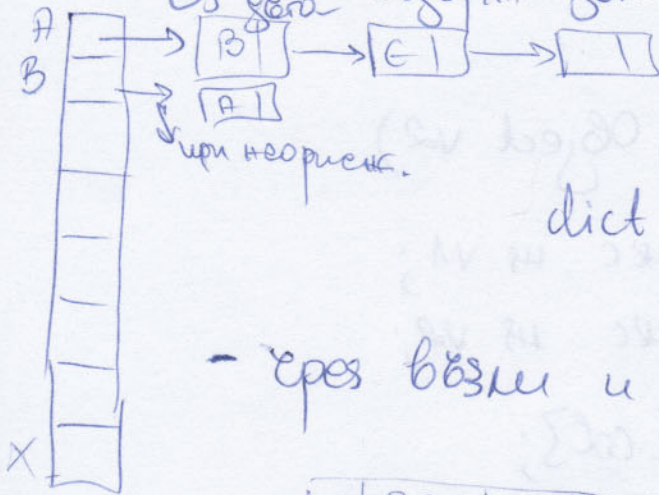
```
        }
```

```
    return list.iterator();
```

```
}
```

Списки на соседа

Списки соседа, хранящихся в блоках



dict (label, list)

- через базис и трансп. разм. соседа (список)

	A	B	C	D
A	1	1	0	1
B	1	1	1	0
C	0	1	1	0
X	1	0	0	1

$$M^2 = M \times M$$

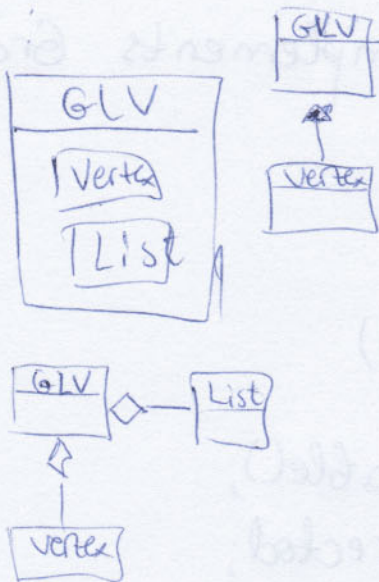
CDuoon

Список на соседей и представление
на рис



GLV {

Vertex vtx;
List adjacencies;



```

class GraphListVertex extends Vertex
{
    protected Collection adjacencies;

    GraphListVertex(Object label)
    {
        super(label);
        adjacencies = new SinglyLinkedList();
    }

    public void addEdge(Edge e)
    {
        if (! containsEdge(e))
            adjacencies.add(e);
    }

    boolean containsEdge(Edge e)
    {
        return adjacencies.contains(e);
    }

    public Edge removeEdge(Edge e)
    {
        return (Edge) adjacencies.remove(e);
    }

    int degree() {
        return adjacencies.size();
    }
}
  
```

```

public Edge getEdge(Edge e)
{
    Iterator edge = adjacencies.elements();
    while (edge.hasNext())
    {
        Edge e1 = (Edge) edge.next();
        if (e.equals(e1))
            return e;
    }
    return null;
}

```

```

abstract public class GraphList implements Graph
{
    protected Map dict;
    protected boolean directed;
}

```

```

    GraphList(boolean directed)
    {
        dict = new Hashtable();
        this.directed = directed;
    }

```

```

    public void add(Object label)
    {

```

```

        if (dict.containsKey(label)) return;
        GraphListVertex v = new GLV(label);
        dict.put(label, v);
    }

```

```

    boolean containsEdge(Object v1, Object v2)
    {
        // no edges v1 to v2 or v2 to v1
        Edge e = new Edge(get(v1), get(v2),
            , directed);
    }

```

```

    return ((GraphListVertex) dict.get(v1)).containsEdge(
public int degree (Object label)
{
    return ((GraphListVertex) dict.get(label)).degree();
}

public Iterator neighbours (Object v)
{
    return ((GraphListVertex) dict.get(v)).adjacencies
        .adjacentVertices();
}

```

```

class List GraphVertex {
    Iterator adjacentVertices()
    {
        return adjacencies.elements();
    }
}

```

```

class GraphListUndirected extends GraphList
{
    GraphListUndirected() { super(false); }

    void addEdge (Object begin, Object end, Object label)
    {
        GraphListVertex v1 =
            ((GraphListVertex) dict.get(begin));
        GraphListVertex v2 =
            ((GraphListVertex) dict.get(end));
        Edge e = new Edge(v1, v2, label, false);
        v1.addEdge(e);
        v2.addEdge(e);
    }
}

```

```

public Object remove (Object label)
{

```

```

    GraphListVertex v = (GraphListVertex) dict.get(label);

```

```

    Iterator v2 = neighbours (label);

```

```

    while (v1.hasNext())
    {

```

```

        Object end = v1.next();

```

```

        removeEdge (label, end);
    }

```

```

    dict.remove (label);

```

```

    return v.label(); // : )
}

```

```

public Object removeEdge (Object begin, Object end)
{

```

```

    GLV v1 = (GLV) dict.get(begin);

```

```

    v2 = (GLV) dict.get(end);

```

```

    Edge e = new Edge (v1, v2, null, false);

```

```

    v2.removeEdge(e);

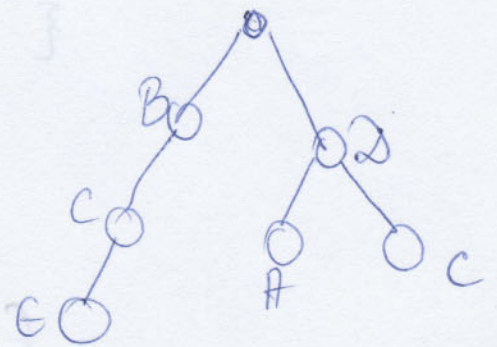
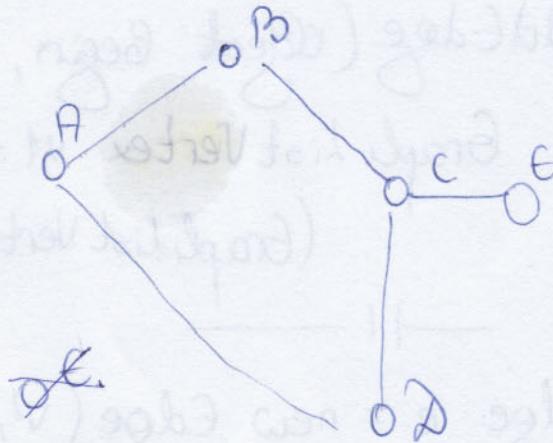
```

```

    return (v1.removeEdge(e)).label();
}

```

Обход и удаление в графе



Начало и конец в графе