E1. (příklad na spojové seznamy) Přepište třídu `Stack` tak, aby se chovala stále stejně, avšak místo pole používala spojový seznam (neomezené délky).

```
class Stack {
    int[] contents = new int[1000];
    int offset = 0;

    void push(int number) { contents[offset++] = number; }
    int pop() { return contents[--offset]; }
}
```

class Elem {

int x;

Elem next;

Elem (int x, Elem next){

this.x = x;

this.next = next;

}


```
class Node {
    int contents;
    Node next;

    Node(int c, Node n) { contents = c; next = n; }
}

class Stack {
    Node head;

    void push(int number) { head = new Node(number, head); }
    int pop() {
        int result = head.contents;
        head = head.next;
        return result;
    }
}
```

class Stack {

Elem head;

- void push (int numb){

Elem e = new Elem (numb, next);

head = e; }

- int pop () {

int result = head.x;

head = head.next;

return result; }

- int size () {

int celkem = 0;

for (Elem e = head; e != null, e = e.next){

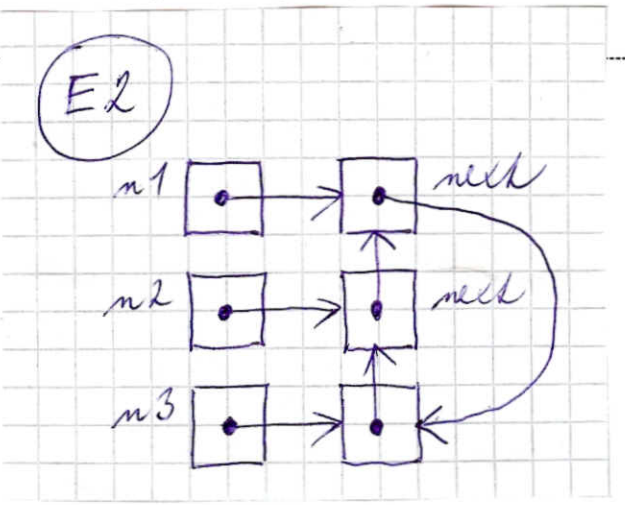
celkem = celkem ++; }

return celkem; }

30.4. /11

E2. (příklad na spojové seznamy) Co vypíše metoda `f`?

```
class Node {
    Node next;

    Node(Node n) { next = n; }
    void set(Node n) { next = n; }
    static void f() {
        Node n1 = new Node(null);
        Node n2 = new Node(n1);
        Node n3 = new Node(n2);
        n1.set(n3);
        for (Node n = n1; n != null; n = n.next) System.out.print("I");
    }
}
```



Cyklí a donekonečna vypisuje „I".

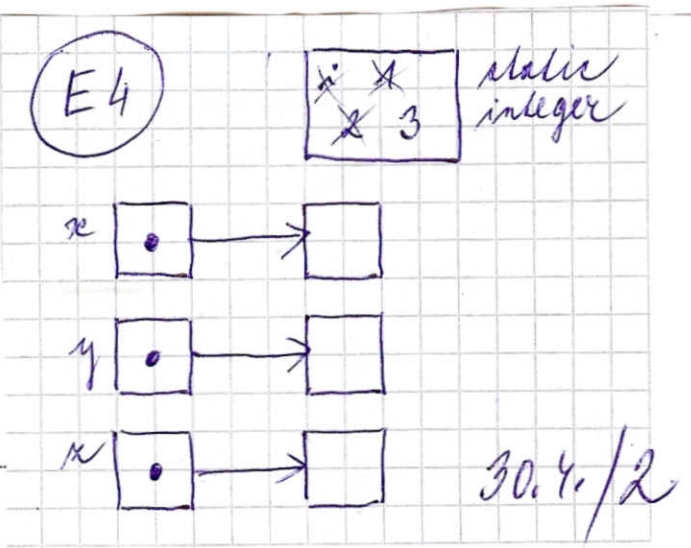
E3. Máte třídu `Rectangle` (reprezentující obdélník) a můžete/musíte ji použít pro vytvoření třídy `Square` (reprezentující čtverec). Použijete skládání nebo dědičnost? Proč a proč ne to druhé?

```
class Rectangle {
    void setX(int x) { /* kod */ }
    void setY(int y) { /* kod */ }
    int getX() { /* kod */ }
    int getY() { /* kod */ }
    void setWidth(int width) { /* kod */ }
    int getWidth() { /* kod */ }
    void setHeight(int height) { /* kod */ }
    int getHeight() { /* kod */ }
}

class Square {
    void setX(int x) { /* kod */ }
    void setY(int y) { /* kod */ }
    int getX() { /* kod */ }
    int getY() { /* kod */ }
    void setWidth(int width) { /* kod */ }
    int getWidth() { /* kod */ }
}
```

Skládání. Dědičnost by v Javě implikovala podtyp a u něj by byl porušen substituční princip.

E4. Co vypíše metoda `f` a proč?

```
class X {
    static int integer;

    X(int i) {
        integer = i;
    }
    int get() { return integer; }
    static void f() {
        X x = new X(1);
        X y = new X(2);
        X z = new X(3);
        System.out.println(x.get());
    }
}
```



Vypíše 3, protože proměnná `integer` je statická.

E5. Napište třídu reprezentující komplexní číslo s operacemi sčítání, odčítání a násobení ([a, b] * [d, e] = [a * d – b * e, a * e + b * d]).

```
class Complex {
    int re, im;

    Complex(int r, int i) { re = r; im = i; }
    Complex plus(Complex c) { return new Complex(re + c.re, im + c.im); }
    Complex minus(Complex c) { return new Complex(re - c.re, im - c.im); }
    Complex times(Complex c) {
        return new Complex(re * c.re - im * c.im, re * c.im + im * c.re);
    }
}
```

E5

class Complex {
dbl a;
dbl ib;
Complex (dbl a, dbl ib){
this. a = a;
this. ib = ib; }

① verze STATIC METODY
- static Complex plus (Complex c1, Complex c2){
dbl a = a.c1 + a.c2;
dbl ib = ib.c1 + ib.c2;
return new Complex (a, ib); }
- static Complex minus ...
- static Complex nasob...

② verze INSTANČNÍ METODY
- Complex plus (Complex c2){
dbl a = this.a + c2.a;
dbl ib = this.ib + c2.ib;
- Complex minus ...
- Complex nasob...

30. 4. /3