E1. (příklad na spojové seznamy) Přepište třídu `Set` tak, aby se chovala stále stejně, avšak místo pole používala spojový seznam (neomezené délky). Spojový seznam naprogramujte od začátku, tzn. nesmíte použít třídy typu `LinkedList` apod.

```
/**
 * Implementace mnoziny, tzn. kontejneru prvku
 * bez vzajemneho poradi a bez opakovani.
 */
class Set {
    int[] contents = new int[1000];
    int size = 0;

    void add(int e) {
        if (!contains(e))
            contents[size++] = e;
    }

    boolean contains(int e) {
        return indexOf(e) != -1;
    }

    int indexOf(int e) {
        for (int i = 0; i < size; i++)
            if (contents[i] == e)
                return i;
        return -1;
    }

    void remove(int e) {
        int i = indexOf(e);
        if (i == -1)
            return;
        contents[i] = contents[--size];
    }
}
```

E1

```
class Elem {
int e;
Elem next;
Elem (...) {...}

class Set {
Elem head;
- boolean contains (int e) {
  for (Elem em = head; em != null; em = e.next) {
    if (em.e == e) {
      return true; } // if // for
  return false; } // contains
- void add (int e) {
    if (contains (e)) {
      return; } // if
    Elem e = new Elem (e, head);
    head = e; }
- void remove (int e) {
    if (head == null) { return; }
    for (Elem em = head; em.next != null; em = em.next) {
      if (em.next.e == e) {
        em.next = em.next.next; } // if // for
      return; }
```

19.5./1

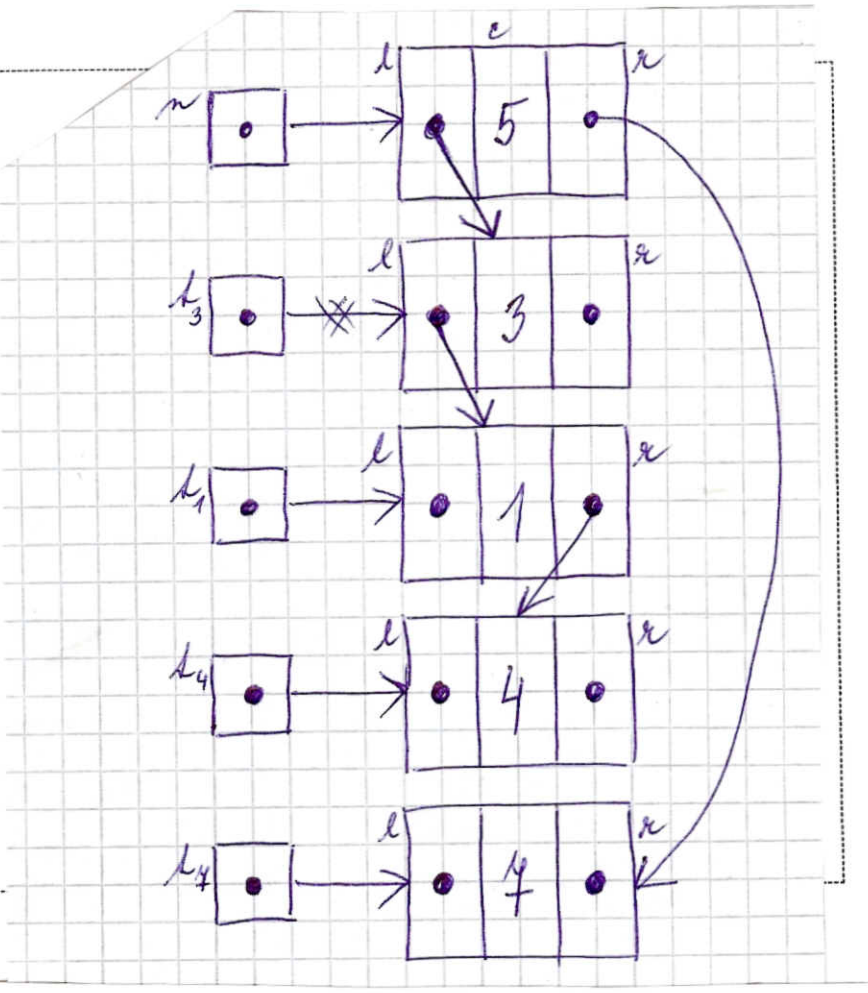
E2. (příklad na spojové seznamy) Nakreslete, jak vypadají objekty v paměti po vykonání metody `f`.

```
class Node {
    int contents;
    Node left, right;

    public Node(int c) {
        contents = c;
    }

    void add(Node n) {
        if (n.contents < contents)
            if (left == null)
                left = n;
            else
                left.add(n);
        else
            if (right == null)
                right = n;
            else
                right.add(n);
    }

    static void f() {
        Node n = new Node(5);
        Node t = new Node(3);
        n.add(t);
        t = new Node(1);
        n.add(t);
        t = new Node(4);
        n.add(t);
        t = new Node(7);
        n.add(t);
    }
}
```



E3. Jedna z metod třídy Set z příkladu E1 by měla být privátní. Určete která a napište proč. Na zdůvodnění si dejte záležet, o uznatelnosti příkladu se bude rozhodovat především podle něj.

metoda „boolean contains()", protože se týká hodnocení pole třídy Set

E4. Změňte metodu `plus` na statickou, zachovejte funkčnost.

```
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);
    }
}
```

static Complex plus(Complex c1, Complex c2){
return new Complex(c1.re + c2.re, c1.im + c2.im);}

19.5./2

E5. Na třídě `Ir` není něco v pořádku. Co?

```
class Kim {
    int contents;

    /* Nastavi hodnotu */
    public void set(int c) {
          contents = c;
    }
    /* Vrati hodnotu */
    public int get() {
          return contents;
    }
}

class Ir extends Kim {
    /* Nastavi hodnotu pouze pokud je parametr zaporny */
    public void set(int c) {
          if ( c < 0) super.set( c);
    }
}
```

Dochází k porušení kontraktu při použití zděděných metod.
Zděděné metody
- musí stejné parametry zachovat
- nebo je může rozšiřovat,
- ale nesmí je zužovat.
- mohou omezovat výjimky.

V příkladu dochází ke „zúžení" dědění metody, což je nepřípustné.

19.5. /3